M I C R O S H E L L

UNIX Features for CP/M

User's Manual Version 2.0

December 9, 1982

New Generation Systems, Inc.
2153 Golf Course Drive
Reston, Va. 22091
(703) 476-9143

COPYRIGHT NOTICE

TRADEMARKS

The following trademarks are referenced throughout this manual:

ACT 80, ACT 86, PASCAL/M, SuperCalc	Sorcim Corporation
BDS C	BD Software, Leor Zolman
CompuPro	Godbout Electronics
CP/M, CBASIC, MAC, MP/M	Digital Research Corporation
Mince	Mark of the Unicorn
Spellbinder	Lexisoft, Inc
Unica	Knowlogy
UNIX	Bell Telephone Laboratories
WordStar, SpellStar	MicroPro International Corporation

How to Run MicroShell without Reading the Manual

Everyone is always anxious to run a new program without having to read through the manual first. Here's how to run MicroShell.

1. **Copy the distribution disk before you do anything else.** It should have the following programs on it:

SH.COM	MicroShell	
SH.OVR	MicroShell Overlay File	(See paragraph 7 below.)
SHSAVE.COM	MicroShell Customization Program	(See Section 5.0)
AUTOCPM.COM	CP/M Patcher for AutoLoad of MicroShell	(See Section 5.0)
D.COM	Expanded Directory Program (Public Domain-Author Unknown)	
MENU.SUB	Sample Menu Shell File	
ERASE.SUB	Sample Shell File	
FULLPRMP.SUB NORMPRMP.SUB	Demonstration shell files which will change the prompt.	

2. With CP/M running in your computer, type "sh", followed by a carriage return of course.

3. MicroShell will sign on with a "B(0) " prompt (or "A(0) " depending on what disk drive you're on.)

4. Give MicroShell a few CP/M commands like "dir" and "stat" to assure yourself that it's working.

5. Now try one of MicroShell's features; type:

```
dir >files
```

You shouldn't see anything but the disk will click as MicroShell puts the directory listing into the file "files". Look at "files" by typing:

```
type files
```

There's your directory! Now let's try another MicroShell feature, multiple commands. Type:

```
dir >>files;type files
```

Now you see a second directory "appended" (the ">>") to the end of the first. And the second half of the command line executed as soon as the first was done!

6. Now try this. Type:

fullprmp

Now you've got a new prompt that includes the disk drive, the user number and a bell (to tell you when a command is done.) (If you got a "fullprmp?", you haven't copied the ".sub" files on the distribution disk to the disk you're using. Go back and do that and then continue.)

If you don't like that prompt, type:

normprmp

and you get the "B(0) " prompt back. You can change the prompt to your liking. See Section 2.4 on "Shell Flags."

7. SH.OVR is the MicroShell overlay file. It is necessary for the following functions:

- Expanded Error Messages. See Appendix B.
- Append (">>") Function. See Section 2.5.1.
- MicroShell Status ("-s") Report. See Section 2.4.8.
- Extended Shell Functions: All functions starting with "%", e.g. %print, %if ...., %iffile ..., etc. See Section 4.9.
- WordStar-like Command Line Editing. See Section 2.8.

If SH.OVR is not present, MicroShell will perform all other functions normally but will respond with "Requires SH.OVR" if one of the above functions is attempted.

7. Now you have the beginning of MicroShell and an idea of the power of the UNIX operating system that it emulates under CP/M. A summary of the special command line characters and shell flags follows. Try each feature to get an idea of what MicroShell can do. This summary is identical to Appendix F and can be removed from the manual and used for reference.

Summary of MicroShell Commands

Special MicroShell Characters in Command Line

Char	Meaning	Example
> >*	Output Redirection OF console Output Redirection OF Printer	stat >filename pip lst:=file >*prntfile
<	Input Redirection	ed file <script
|	Pipe output to input of next cmd	prog1 | prog2 | ... stat *.*|pip lst:=con:
^	"^" in shell and input files causes next character to be its control equivalent.	^C (or ^c) changed to 03
: ;	When first character on a shell file line, causes line to be treated as a comment (ignored).	: this is a comment ; this is a comment too
+	Echo redirected Output to Console	stat >+filename prog1 |+ prog2 | ...
-	Return "character ready" to console input status calls	sysgen <-script prog1 |- prog2 | ... prog1 |+- prog2 | ...
;	Separate commands	era *.bak;stat;ed test <script
"	Treat arguments with embedded spaces or tabs as 1 argument and ignore special characters inside quotes	echo "This is one argument"
\	Ignore special meaning of next character	dir \>file (filename ">file")
$	Argument substitution in shell (command) files (0-18)	copyfile test if "copyfile.sub" contains: pip b:=a:$1 then MicroShell executes: pip b:=a:test
!	WordStar-like Command Line Editing	B(0) mustake!
$T $t	Redirect Input back to console in a shell file	ddt <$T
$P $p	Redirect Output to Printer	dir >$P

Shell Flags

Flag	Meaning	Example
+f or +F (Default)	Auxiliary file search enable	B(0) +F (Auxiliary file search on)
-f or -F	Auxiliary file search disable	B(0) -F (Auxiliary file search off)
+g or +G (Default)	Gobble line feeds during Input redirection	B(0) +G (Line feeds removed from input)
-g or -G	Don't gobble line feeds during Input Redirection	B(0) -G
-l or -L or +l or +L	Login current disk (after changing disks)	B(0) -l B(0) (new disk logged into CP/M for writing)
-p or -P or +p or +P	Prompt string Uses "C"-like format: % - Next char special (%% gives %) n/N - Newline (CR, LF) d - Lower case drive D - Upper case drive u/U - User number	-p "%n%D(%u) " gives: (CR, LF)B(0) -p "%nDrive:%D User%U %" gives: (CR, LF)Drive:A User:0 %
-s or -S or +s or +S	Shell Status report (Shows status of flags)	B(0) -S [displays status report]
+u or +U (Default)	Upper case translation of command line (like CP/M)	B(0) +U B(0) echo this is upper case THIS IS UPPER CASE
-u or -U	No case translation on command line (allows passing lower case command line to a program)	B(0) -U B(0) echo this is lower case this is lower case
+v or +V	Verbose mode: Echo commands before execution	B(0) +V (All commands echoed) comfile test data pip b:=a:test pip b:=a:data
-v or -V (Default)	Disable Verbose mode	B(0) -V (No echo of commands)
-x or -X	Exit MicroShell and return to CP/M	B(0) -x B>

Additional Flags: D - Section 2.4.2, M - Section 2.4.6, T - Section 2.4.9

Table of Contents

Table of Contents (Cont)

1.0 Overview

MicroShell is a CP/M program which adds powerful, user-friendly capabilities to the CP/M operating system similar to many of the functions available in the UNIX operating system. Compatibility with existing CP/M software is retained while adding the UNIX features to the operation of existing CP/M software. New software applications and tools can be designed and implemented with much less effort using the features available in MicroShell. MicroShell can be tailored to a user's system and experience level, providing additional information and help for a new user or crisp, elegant power for an experienced user.

1.1 System Requirements:

MicroShell requires CP/M 2.2, at least 32 K of memory and at least one disk drive.

MicroShell requires 8K of system memory when loaded, residing directly below CP/M. A 64K CP/M computer will thus appear to be a 56K system to a program running under MicroShell. As most commercial software has been written to run under MP/M, where only 48K is normally available to user programs, this memory requirement does not normally restrict the operation of a program.

The minimum disk space required by the MicroShell program and its overlay is 20K, although once loaded, neither MicroShell nor its overlay are necessary for most of MicroShell's functions.

1.2 Summary of MicroShell Features:

The new user of MicroShell should first read the Preface -- How to Run MicroShell without Reading the Manual, following the examples by actually using MicroShell. After he has gained an initial familiarity with MicroShell, Basic MicroShell Commands, Section 2.0, will provide more details on using MicroShell.

2.0 Basic MicroShell Commands

This section will cover the basic MicroShell Commands, including:

2.1 - Executing MicroShell
2.2 - CP/M-like functions
2.3 - Forming MicroShell command lines
2.4 - MicroShell Flags
2.5 - Output Redirection
2.6 - Input Redirection
2.7 - Pipes
2.8 - Command Line Editing

2.1 Executing MicroShell

The MicroShell program is named "sh.com" and is executed by typing:

```
sh

or

sh [ initial command line ]
```

from the CP/M prompt ("A>", if the user is logged into CP/M on the A disk drive.) MicroShell will respond with:

```
MicroShell Version X.X
Copyright (c) 1982 New Generation Systems

(If an initial command line was given, the command
 is executed prior to the initial "A(0)" prompt.)

A(0)
```

The default MicroShell prompt displays the drive letter in upper case followed by the current user number in parentheses. The user may customize the prompt as desired. See Section 2.4.7.

The initial command line may be any legal MicroShell command. For example, if the user wanted to log onto drive B immediately, he could execute MicroShell with the following command line:

```
sh b:
```

MicroShell would then change the default drive to B prior to displaying the initial prompt. If a user desired to set some of the shell Flags (Section 2.4) on initial entry, a shell file (Section 4) could be executed on initial entry to perform these functions. If the file, "init.sub" contained the following lines:

```
-V
b:
dir
```

then by typing:

sh init

MicroShell would turn off the Verbose Flag (Section 2.4.11), change the default drive to B, and display the directory prior to issuing the initial prompt. The shell Flags and shell files are fully described in Sections 2.4 and 4 respectively.

Before we proceed to discuss MicroShell's other features, here's how you get out of MicroShell and back to CP/M. Just type the -x (or -X) "eXit" shell Flag, by itself, in response to the MicroShell prompt. MicroShell will exit back to CP/M and CP/M will display its prompt for the current drive, e.g.:

```
A(0) -x      (<---user types "-x", carriage return)
A>           (<---CP/M prompt)
```

2.2 CP/M-like Functions

MicroShell, on initial entry, relocates itself just below the CP/M Basic Disk Operating System (BDOS), replacing the CP/M Console Command Processor (CCP). The CCP is the part of CP/M which interprets commands by the user and performs the built-in functions: DIR, ERA, REN, SAVE, TYPE and USER. The vast majority of user programs - word processors, compilers, accounting programs, etc. - actually overlay the CCP to use its space in memory for their own use. MicroShell replaces (overlays) the CP/M CCP but does not permit user programs to overlay MicroShell, as it must remain present to perform many of its functions. All of this action is invisible to the majority of CP/M programs which use CP/M's design entry points to accomplish their functions. Some programs, mostly older ones, "look around" inside of CP/M to perform some function. This is "not cricket" from a software compatibility and transportability aspect and these programs may not operate properly with MicroShell; they also probably won't work under new revisions of CP/M. See Appendix C for a list of known compatible and incompatible programs.

Since MicroShell overlays the CP/M CCP, it performs all of the functions that the CCP normally performs. The following commands will be executed by MicroShell just as the CP/M CCP would execute them, except as noted:

CCP Command	Action	MicroShell Differences
DIR [afn]	- Directory listing	"-s" option for "SYS" files
ERA [afn]	- Erase file(s)	No "*.*" warning
REN newname=oldname	- Rename a file	No "_" alternate separator
TYPE filename	- Type a file	"TYP" paged variation
SAVE nn filename	- Save "nn" 256-byte pages of memory beginning at 100 Hex	
drivename:	- Change default drive	
user NN	- Change user area to #NN	Allows access to areas 0-31. CP/M only allows access to 0-15 while 16-31 can only be accessed from a user program call to CP/M.)

where: "afn" is "ambiguous file name" e.g. "*.*"

We will assume that the MicroShell user is familiar with these CP/M commands. The new CP/M user may wish to review these commands in the Digital Research CP/M documentation.

The CP/M CCP also provides the user with certain line editing functions to change a command line which has been entered in error prior to executing it (e.g. backspace, delete, control: u, x, r, e, etc.) These line editing features are supported by MicroShell. Again, the new CP/M user may wish to review these commands in the Digital Research documentation. MicroShell also provides an enhanced, WordStar-like, command editing capability which can be used to edit the current or previous command line. This facility is described in detail in Section 2.8.

MicroShell provides a variant of the CP/M "TYPE" command: "TYP". "TYP" may be used to type a file with automatic pauses after each pageful. The default setup displays 23 lines of the file and then waits for the user to type any character before proceeding to display the next page. The user may use control C during the pause to abort the rest of the display. The 23 lines/page setting is customizable by the user for other than the standard 24 line terminals. See Section 5.2.

MicroShell also provides a variant of the CP/M "DIR" command to display "system" files:

```
B(0) dir -s
```

will display all files including those marked (by STAT) as system files. Or:

```
B(0) dir -s *.com
```

will display all ".COM" files, including "system" files.

"Wild-card" characters "?" and "*" may be specified in any of the "CP/M-like" commands. In standard CP/M, only "era" and "dir" will allow these characters. MicroShell permits their use anywhere a file name is specified. MicroShell's response to a "wild-card" character is to use the first file name encountered in the disk directory which matches the file specification. Thus for example, if you type:

```
B(0) type ab*
```

MicroShell would "type" the first file name it found that started with "ab", e.g. "abzzt" or "abstract", etc. Notice that this feature does <u>not</u> cause an expansion of the action to multiple files. Although this feature can save some unnecessary typing, some care must be used with it. As an example, if you type:

```
B(0) *
```

MicroShell will execute the first ".com" file it finds in the directory, or even more exciting than that, if you type:

```
B(0) save 2 *.*
```

MicroShell will first <u>erase</u> all files in the directory before "saving" 2 256-byte pages into a file named "????????.???" !! This is a fairly unusual

hypothetical situation which does not in practice cause any problems but the user should be aware of the feature.

A final CP/M function which is also provided by MicroShell is control P which toggles the printer on or off.

With the exception of the differences dicussed above, all CP/M CCP functions execute identically under MicroShell and provide the same error messages as the CP/M CCP on encountering an error condition.

Some users have asked why MicroShell doesn't give an error message if a program is run which is too large to fit under MicroShell. It does! The same error message which the CP/M CCP issues -- "BAD LOAD" -- will be displayed by MicroShell if an attempt to load a program which is too large to fit under MicroShell is attempted. Here's the rub. Most programs, in addition to the memory space they need for the program itself, need some space between themselves and CP/M for data and the stack. Neither MicroShell nor CP/M can predict how much space this will be. So, it is possible for a program to successfully load under MicroShell, but then not be able to run properly due to insufficient data space. Many programs gracefully check the available space and exit with an error message in this case. Some just crash! At any rate, MicroShell responds like plain CP/M would have in the same situation.

Another similar question we get occasionally is whether there is any way to make MicroShell "just a little bit" smaller. With the distribution version of MicroShell, the answer is "no"; overlays are already in use for all of the functions which we feel can be allowed to require a disk access and there is no run-time facility for decreasing MicroShell's size. Neither is it likely that future versions of MicroShell will be substantially smaller than 8K. MicroShell is now completely written in assembly language with all the "tricks" we know of to minimize memory size. It is possible for us to create a "custom" version of MicroShell for an individual user with a special need. We have done this in the past for a few customers. Our charge depends on the effort required but is priced on a time basis. Contact us if you have such a need. We would emphasize, however, that creating huge applications programs may cause future problems. If there is any possibility that your program will be required to run in a multi-user environment in the future, many MP/M configurations provide only a 48K memory space for user programs. Also, your operating system may require additional memory when you upgrade to a hard disk, or when you upgrade to a new version of CP/M. So before you exceed 48K we would recommend that you segment your program and use MicroShell's menu features (Section 4.9) or use overlays for your program.

2.3 Forming MicroShell Command Lines

The process of entering a command to MicroShell is identical to CP/M, i.e. the main command or program name is entered first, followed by any arguments the command requires. For example:

```
stat *.*
```

"stat" is the main command (program) to be executed and "*.*" is an argument to "stat" telling "stat" to display a list of all files in the current directory (actually the current disk drive and user number.) MicroShell

accepts an identical command. In addition, however, MicroShell permits additional arguments to be given to perform special MicroShell functions. For example, if the user typed:

```
stat *.* >statout
```

MicroShell would send the normal output of the "stat *.*" command to the file "statout" instead of displaying it on the console. This is called output redirection and is one of the UNIX features brought to CP/M by MicroShell. It is discussed in detail in Section 2.5. MicroShell's other features, input redirection, pipes, multiple commands on one line, etc., are similarly invoked.

The maximum length of a command line to MicroShell is 84 characters.

A note is in order here about how MicroShell works. In the previous example, "stat *.* >statout", MicroShell recognizes the ">statout" portion of the command, strips it from the command, finds and loads the file "stat.com". "stat" never knows that ">statout" was typed. It sees the command just as if "stat *.*" was typed from CP/M! The other special MicroShell commands are similarly stripped out of the command line by MicroShell before it is passed to the "main command".

White Space in Command Lines: In general, "white space" (spaces or tabs) may be used freely in commands to MicroShell. In particular, "white space" is optional before and after redirection symbols ("<", ">"), pipe symbols ("|") and semicolons. One place there cannot be "white space" is between a redirection symbol or pipe symbol and a "+" or "-" modifier. For example: "stat *.* >+ out", "stat *.*>+out" and "stat *.* >+out" are all legal command lines, but "stat *.* > + out" is not.

The other MicroShell special commands are covered in detail in the following sections and summarized in Appendix F.

2.3.1 Escaping MicroShell Special Characters:

The characters ">", "<", "|", ";" , "\", "!" and double quotes (") have a special meaning to MicroShell when used anywhere in a command line. Two of these special characters can be permanently changed if desired by the user: ";" -- the multiple command separator and "!" -- the command line editing character. See Section 5.2 for a discussion of how to change these characters.

If it is desired that one of the other special characters be used in a command line for non-MicroShell functions (i.e. the user program needs them) their special meaning may be ignored by MicroShell in one of the following three ways:

1. Precede the special character with an "\", i.e.:

```
save 1 \>file
```

will create a filename ">file". To get a "\" to be passed through MicroShell, type two "\" 's, i.e.:

```
save 1 \\
```

will create a file named "\".

2. Enclose the entire argument containing the special characters in double quotes ("), i.e.:

```
save 1 ">><|"
```

will create a file named >><|. (Note that quotes (") cannot themselves be enclosed in quotes but must be escaped with a \.)

3. During input redirection, special character meaning may be escaped by setting the Transparency ("+T") Flag on. Using this Flag has some other effects which should be studied in Section 2.4.9.

The characters "+" and "-" have a special meaning after ">", "<" or "|" and "#" has special meaning after ">". These characters must be escaped with a "\" if the user really wants to have the "+" or "-" as part of the file or command name. It is best to stay away from the special characters; they're not common in file names anyway.

The dollar-sign character ("$"), followed by a number from 0 to 18 has special meaning in a shell file (argument substitution.) This special meaning can be escaped by using double dollar-signs -- $$ -- (to be compatible with CP/M)

Three other characters may be assigned special meaning in input files: %exchar, %tchar and %schar. See Section 4.9.4 for the meaning of these characters and how to assign them.

To round out the discussion of special MicroShell characters, all of the shell Flags (Section 2.4), when used by themselves on a command line, are recognized by MicroShell as special commands and cannot be escaped.

2.3.2 Multiple Commands on a Line:

MicroShell will accept multiple commands on one line with each command separated by a ";". For example, the MicroShell command:

```
era *.bak;dir;stat
```

is equivalent to giving the following command sequence to CP/M:

```
A> era *.bak
A> dir
   ---------------------------------------------
            CP/M displays directory
   ---------------------------------------------
A> stat
   ---------------------------------------------
          stat displays space remaining
   ---------------------------------------------
```

```
A>
```

The only difference is that the MicroShell user can give all three commands at once and watch them sequentially execute! He doesn't have to wait for each command to complete before entering another command. The benefit of this feature, which comes from the UNIX operating system, is that we as human beings think in terms of logical processes which may involve numerous commands to accomplish. If we can type in all of the necessary commands to carry out a logical process, we are then freed from that level of detail and can concentrate on the results or on the next process. Thus the computer and its software has done what it does best - keep track of multiple and/or repetitive tasks while the user is free to think about logical processes. In Section 4 we will cover one higher step in this same philosophy: putting a number of individual commands into a file -- a shell file -- and executing them by merely typing the name of the file.

Limitations on Multiple Commands:

There are several restrictions on the use of the multiple command feature of MicroShell:

1. The maximum command line length of 84 characters cannot be exceeded. The total command (i.e. all characters entered up to the carriage return) is counted - not just each individual, semicolon-separated command.

2. Only 17 arguments are permitted for each semicolon-separated command. That is: a main command and 17 arguments to the main command.

3. Any commands on a line after the MicroShell eXit Flag "-x" are ignored, since MicroShell relinquishes control to CP/M after the "-x" Flag is executed.

4. Any redirection specified in one command is not carried over to the next semicolon-separated command. For example:

```
stat *.* >statout;dir
```

will cause only the "stat" output to be redirected to the file "statout", while the directory display from "dir" will come to the console as it normally would. This is not really a restriction; it is what would logically be expected to happen. It is mentioned to clarify the scope of action of redirection.

5. Some programs use the semicolon as a special character themselves. Provision is made to change the character MicroShell uses for multiple command separation. See Section 5.2.

2.3.3 Interrupting a MicroShell Command:

If a series of commands are given on one line, previous versions of MicroShell provided a means of interrupting the sequence of commands. This mechanism was somewhat unreliable as it required typing the escape character and having MicroShell "see" it in between commands. Many programs "ate up"

any input characters, removing the escape character before MicroShell could "see" it. So this mechanism has been removed from the current version except for during shell file execution (see Section 4.7.) The only possible way to stop a series of semicolon-separated commands now is to "freeze" the output with control S (obviously the program must be doing some output for control S to stop it) and then typing control C. This normally will abort the entire command line. Note that any abort mechanism built into a program, such as control C with the BDS C compiler, will still abort the program, but MicroShell will then execute the next logical command in the line.

2.4 MicroShell Flags

Several of MicroShell's features can be altered by the user from within MicroShell itself. These commands to MicroShell are called Flags. The following is a summary of MicroShell's Flags and their default settings. All default settings can be modified by the user. See Sections 4.9.4 and 5.1.

Summary of MicroShell Flags

Flag	Meaning	Default *
D	Delays	0
F	File Search (On/Off)	On
G	Gobble line feeds (On/Off)	On
L	Login disks	-
M	Mode - End of File (+=CP/M,-=UNIX)	CP/M
P	Prompt change	A(0)
S	Status report of Flags	-
T	Transparency Character Flag	Off
U	Upper case command line	On
V	Verbose mode: command echo	On
X	eXit MicroShell to CP/M	-

* Default values with "-" indicates Flag performs action only and has no state.

2.4.1 Issuing a Flag Command:

A Flag command is issued as a normal command to MicroShell, either on initial startup, or in response to the MicroShell prompt, or in a shell file or as one of multiple (semicolon-separated) commands on a line. The Flag is preceded by a "+" or a "-" depending on whether the user wants to turn the feature on (+) or off (-). Flags may be issued in upper or lower case.

For example:

A(0) -F<carriage return>

turns off the automatic file search feature while:

A(0) +V<carriage return>

turns the Verbose mode on. If there is not an "On/Off" function associated with a Flag, as with the "S" status report Flag, either a "+" or a "-" may precede the Flag. The following command line is legal:

A(0) +V;-L;-U;-S<carriage return>

and would turn on the Verbose Mode, login all disks again, turn off Upper case command line translation and print a status report of the new Flag conditions.

2.4.2 D Flags: Delays Default: 0

The Delay Flags provide a method for specifying some built-in delays in MicroShell. There are 4 delays. Their syntax, meaning and range are:

Syntax	Delay Set	Range of Delay
-D 1 NNN	Input Character Delay	0 - 999 tenths of seconds
-D 2 NNN	Input Line Delay	0 - 999 tenths of seconds
-D 3 NNN	Output Prompt Delay	0 - 999 tenths of seconds
-D 4 NNNNN	Input Ignore Factor	0 - 65535 "factor"

The first 3 delays are in 0.1 seconds, i.e. a setting of 100 will provide a delay of approximately 10 seconds. The input delays affect character and line inputs respectively from an input or shell file. Their purpose is to permit slowing down a self-running demonstration created with MicroShell by inserting a delay whenever the program asks for input.

The third delay is executed just before MicroShell displays the command prompt. It is provided mainly for one situation: Heath/Zenith computer users using the PIE editor. Apparently the PIE editor issues a terminal reset command on exiting which does not have time to complete resetting the terminal before MicroShell tries to display the prompt. The result is that several characters from the prompt string are lost before the terminal is ready to accept characters again. Setting this delay to 1 will solve the problem.

The fourth and final delay is the Input Ignore Delay. Its use is rather involved. If input is redirected to a program which does a lot of things between getting characters, it is possible for MicroShell to send characters to the program so fast that the program never has a chance to do whatever it's supposed to do. It spends all its time getting characters. In some cases, the program's input buffer is even filled up causing the problem to bomb. SuperCalc, and Mince are examples of this. To solve this problem, this Delay Flag allows setting an "Input Ignore Factor" in MicroShell. What this does is to supply a character to the program only once for every "X" times the program "asks" if a character is ready. "X" is the Input Ignore Factor. So if it's set at 500 (which is a good value to start with) then only once in every 500

times the program asks if a character is ready, will MicroShell return "Character Ready" status. The effect of this is to make shell files and input redirection work fine with these types of programs. Additionally, it provides a way (in addition to the other Delay Flags) to set the effective input rate to the program.

Make sure that this Flag is set back to 0 when you don't need it or all your shell files and input files will run much more slowly!

2.4.3 F Flag: File Search (On/Off) Default: On

The "F" Flag turns MicroShell's automatic file search feature on (+)(default) and off (-). The automatic search features are explained in Section 3. The F Flag only affects file search not command search. If the user does not want MicroShell to search for files that match MicroShell-responsible extensions (.SUB, .OV?, and .COM in the as-delivered MicroShell), the command:

```
A(0) -F
```

will turn automatic file search off. It will remain off until the user either turns it back on with:

```
A(0) +F
```

command or MicroShell is restarted from CP/M (if the user hasn't changed the default F Flag condition.)

This Flag is useful in cases where a program might be trying to erase or rename a file and the user doesn't want that to happen.

The automatic file search feature is really provided to permit a program to read files - like Wordstar reading its .OVR files or CBASIC loading .INT files, etc. When programs are erasing, rewriting or renaming files which have MicroShell-responsible file extensions, the user should be very careful about using the automatic file search feature. Be sure you know what's happening before files are erased or changed forever.

Two common programs fit exactly this category, where automatic file search is not desired: CP/M's PIP file copy utility and several available programs named COPY. So MicroShell explicitly recognizes when the user has invoked these two programs and turns the F Flag off. When PIP or COPY is complete, MicroShell restores the F Flag to its previous condition. There are no other programs we've come across which require this Flag off.

2.4.4 G Flag: Gobble Line Feeds (On/Off) Default: On

During input redirection, MicroShell may be reading a file of commands into a program -- for example an editor script. When the file of commands is created with most editors, a carriage return followed by a line feed is placed at the end of each line.

Now comes the confusing part. Programs can ask CP/M for input

characters from the keyboard in two ways; they can ask for one character at a time (BDOS call 1 or 6) or a line of characters until a carriage return is typed (BDOS call 10.)

Now consider our file which we want to use as input. If the program calls for a line of input characters, MicroShell will supply successive lines from the file, automatically stripping the carriage return and line feed from the end of each line. Since the program called for a line of input, MicroShell knows that it shouldn't send the carriage return and linefeed. This is the most common call for input characters. DDT takes its input from this type of call, as does the command mode of the CP/M editor, ED. So everything works fine.

On the other hand, if the program calls for its input one character at a time, the linefeed after each carriage return in a file is probably going to confuse most programs, since the program is probably expecting the user to type a line of input followed only by a carriage return. Leaving the linefeed out of the script file is one solution; but when the file is "type"d, all the lines appear on top of each other, distinctly confusing the user. Having MicroShell always throw away linefeeds after carriage returns is an alternative solution. This works fine for most programs, like CP/M's "ED" in the insert mode and most editors. But what if you wanted to write a program that took each character of a file, did something with it and passed it on to the output? This type of program is called a filter and is frequently used in the UNIX environment. If you want MicroShell to do the file handling for you, redirecting input and output so you can write a simple keyboard to screen filter, then you don't want MicroShell eating up the linefeeds!

So after all this explanation, MicroShell has a Flag to selectively gobble linefeeds on single character input -- you choose. Note that MicroShell always gobbles a linefeed after a carriage return when a program asks for a line of input characters; you can't change that (and probably wouldn't want to.)

To try to solve this nuisance for a part of the cases which might occur, MicroShell automatically sets some Flags when a pipe is invoked. See Section 2.7.1 for a discussion of how this works.

If you find you want the Gobble Flag off most of the time, if you're using filters a lot with input redirection instead of editor scripts, you can change the default using SHSAVE. See Section 5.1. Also remember that you can make up a shell file which sets this Flag, executes some command and then restores the Flag.

2.4.5 L Flag: Login Disks

The L Flag is provided to login a new disk, if disks are changed after MicroShell is started. If a disk is changed after MicroShell has accessed it once, then CP/M logs the disk as Read/Only and any attempt to write on the disk will result in a BDOS error.

Issuing the L Flag (either +L or -L or +1 or -1, there's no difference), will cause MicroShell to login drive A, login the disk on the drive the user is currently on (the default drive) and reset all other disks in the system.

Then if the user selects a new drive, CP/M will log it in as read/write.

For example, if the user is currently on drive B and wants to put a new disk in drive B, he performs the following steps:

1. Change the disk in drive B.
2. Type -L or +L followed by a carriage return.

The head will load on drive A to login A and then on B to login B, (since B was the default drive.) Both drives A and B are now logged back into CP/M for read and write. If another drive is selected after the -L, CP/M will log it in again, since it "forgets" all disks logged in prior to the -L command.

All of this happens without the warm start that usually happens in CP/M. This is good and bad. Good because it's quicker. Good because on some double density systems it permits a single density disk without CP/M on the system tracks to be logged into drive A, permitting for example, PIPing between two single-density disks. Why bad? Bad because in some double density systems, density selection only occurs during a real warm start. On these systems, MicroShell must be exited to log in a different density disk. Try your system to see what it requires. The CompuPro system we use will properly determine disk density within MicroShell.

Control C typed at the MicroShell command prompt is identical to issuing the -L Flag. If you change disks and forget to log in the new disk before issuing some command, you may get a BDOS error. Just respond with a Control C. MicroShell will login the disk and you can execute the command again with the command line editing feature. See Section 2.8.

2.4.6 M Flag: Mode at End of File Default: CP/M

The Mode Flag determines what happens when the end of an input file is reached when using input redirection, reading input from a shell file or using a pipe. If the Mode Flag is set for CP/M mode ("+M"), input will revert to the keyboard after all characters have been used from the input file as would happen if a CP/M SUBMIT file with XSUB was used under CP/M. Normal operation of the program from the keyboard then continues. This is an excellent way to give a program some initial commands and then enter an interactive mode. With WordStar, for example, the initial commands could set up various WordStar features such as margins, and then allow the user to have control of the input. There is even a provision made to take some input from an input file, shift the input to the keyboard, and then shift back to the input file. See Section 4.9.4.

The UNIX setting of the Mode Flag ("-M") is designed for use with programs which expect a control Z followed by a carriage return, at the end of the input file. These are typically "filters", programs which read a stream of characters, performing some action on them, and then exit at the end of the input stream. UNICA is an example. In the UNIX mode, when MicroShell detects the end of the input file (a control Z or no more sectors in the file), a control Z followed by a carriage return is passed to the program. The program must recognize the control Z-carriage return and then terminate by one of the methods discussed in Section 2.6.2. If the program calls for more input after MicroShell has sent the control Z-carriage return, MicroShell will force an

exit from the program.

The Mode Flag is normally set automatically by MicroShell to the UNIX mode when a pipe is invoked. See Section 2.7.1 for a discussion of MicroShell's setting of the Flags for a pipe and Section 5.2 for how to override the automatic setting feature.

2.4.7 P Flag: Prompt Change Default: "A(0) "

The P Flag permits the MicroShell prompt to be changed from within MicroShell. The following are examples of possible prompts:

```
A(0)                          (default)

Drive: A User:0
>

A(0)(bell)                    (bell sounds after prompt)

On Drive:A User:0             (Wordy!! but possible)
Enter command:
```

The flexibility for setting the prompt to match the user's needs is available. Short prompts for experienced users; long for occasional users; bells for users who do something else while their programs run!

This Flag has an argument: the new prompt string. For example, the proper command to get the default prompt (back) is:

```
-p "%n%D(%u) "      (Note the ending space)
```

Well, here are the syntax rules for the prompt string. The syntax is a rough takeoff from the formatted print syntax for the C programming language.

Character	Meaning
%	Special character: next character means something special
N or n	New line: carriage return, line feed. %n means substitute a carriage return and linefeed. Note that unless you want a really unusual display, most prompt strings should begin with %n.
D	Upper case drive letter. %D substitutes the currently-selected drive in the prompt string.
d	Lower case drive letter. %d as above.
U or u	User number. %u substitutes the currently-selected user number into prompt string.

Since "%" has special meaning, to get a "%" to print in the prompt string, enter two "%"'s - %%. If embedded spaces or tabs are desired in the prompt, enclose the whole prompt string in quotes, like we did for the default prompt. If a bell or some other control character is desired in the prompt, just type the actual control character into the prompt string. Alternatively, if you do not enclose the prompt string in quotes, you may use the up-arrow, letter combination (e.g., ^G for the bell.) Note that if you do not enclose the prompt string in quotes, embedded spaces or tabs are not permitted.

Now you can build your own prompt. You can put the prompt command in a shell file and execute it when desired by typing the name of the shell file. Look at our examples, "normprmp.sub" and "fullprmp.sub", on the distribution disk. Or you can permanently change the default prompt: see Section 5.1.

Maximum Prompt Length: 40 characters.

Note that with an intelligent terminal, some really unusual displays can be generated. For example, you could always have the prompt clear the screen, or change pages.

One word of warning: don't end your custom prompt string in a single "%". The prompt processing routine does not handle every possible error. A single ending "%" would mean that the next character was special and since there is no "next" character, a real problem can occur. Also remember to include the prompt string in double quotes if it has any spaces, tabs or special characters in it. It's easiest just to use double quotes all the time.

2.4.8 S Flag: Status Report

The S Flag provides a display of the status of various MicroShell features, including the Flags. The S Flag can be issued as +S or -S (or +s or -s). The report looks like this:

```
+-F             File Search:                          On
+-G             Gobble Line Feeds:                    On
+-M             Mode - End of File(+=CP/M,-=UNIX):CP/M
+-T             Transparency Character Flag:          Off
+-U             Upper Case Command Translation:       On
+-V             Verbose: Echo shell files:            On
+-P "string"    Prompt String: %n%D(%u)
-D 1 NNN        Input Character Delay:    0
-D 2 NNN        Input Line Delay:         0
-D 3 NNN        Output Prompt Delay:      0
-D 4 NNNNN      Input Ignore Factor:      0
%path  "str"    Disk Drive Search Path:A
%ftype "str"    File Types-Auto Search:SUB OV? COM
%shext "str"    File Type -Shell Files:SUB
%sdrive char    Scratch Drive-Shell Files: A
%clrstr "str"   Screen Clear String:       ^Z
%break on/off   Break during Shell Files:  On
%bchar  char    Break Character:           Esc
%exchar char    Program Exit Character:    None
%tchar  char    Revert to Keyboard Char:   None
%schar  char    Revert to Shell File Char: None
```

The first 11 lines in the display are the MicroShell Flags and the remaining lines which begin with "%" are extended shell commands. See Section 4.9 for an explanation of the extended shell commands.

The status display is one of the features which requires the SH.OVR overlay file. If MicroShell can't find the overlay file it will sound the bell and display the error:

```
B(0) -s
Requires SH.OVR
B(0)
```

2.4.9 T Flag: Transparency Character Flag (On/Off) Default: Off

During input redirection and pipes (a special case of input redirection), certain characters in the input file have special meaning:

\	Take next character literally (ignore special meaning)
^	Make next character a control character
CR	Remove following LF if Gobble Flag is on
Control Z	End of input file
%exchar	Program exit character (if set - see Section 4.9.4)
%tchar	Revert Input to terminal character (if set - see Section 4.9.4)

If the Transparency Flag is turned ON, all of the above special character meanings are ignored, i.e. every character is sent to the program without any interference by MicroShell. MicroShell will only take end of file action at the physical end of the last sector of the input file (which may be up to 127 characters after the end of the text in the file, i.e. after the control Z.) So the program must handle end of file recognition if you want to stop reading when a control Z is read. Note that this mechanism allows passing object code files through a filter if you want.

With the Transparency Flag on, MicroShell's action at the physical end of file (the end of the last sector in the file), still depends on the state of the M (Mode of End of File) Flag. In the CP/M mode, input will revert to the keyboard; in the UNIX mode, a control Z will by sent followed by a carriage return. (See Section 2.4.6 above.)

Note that the state of the Gobble Line Feed Flag has no effect if Transparency is ON.

MicroShell automatically handles turning ON the Transparency Flag when a pipe is invoked by the user unless overriden. See Section 2.7.1.

CAUTION: If you turn the Transparency Flag ON in a line of a shell file, it must be turned off later in the same line or MicroShell may not be able to read the lines from the shell file properly. As an example, the following shell file will work properly:

```
; sample filter shell file with explicit input redirection
+t;filter <file1 >file2;-t
nextcmd ....
```

In a pipe, MicroShell takes care of the Transparency Flag:

```
; sample filter shell file with pipes
type file1 | filter >file2
nextcmd ....
```

2.4.10 U Flag: Upper Case Command Line (On/Off) Default: On

The U Flag permits the command line passed to a program to either be in the CP/M-compatible upper case only mode or in upper/lower case as the user types it. (This is the command line which is set up at 80 Hex by the CP/M CCP or MicroShell prior to executing a program.)

Regardless of the state of this Flag, MicroShell translates all file names (i.e. the "main command" name and the setup of the secondary file control blocks at 5C and 6C Hex) to upper case to remain compatible with CP/M upper case file names.

It is often desirable to pass a program upper and/or lower case arguments. Issuing the -U Flag will permit this. Note, on the other hand, that some programs will not like to see lower case arguments, so use this capability with caution.

2.4.11 V Flag: Verbose Mode: Command Echo(On/Off) Default: On

The V Flag permits the user to have MicroShell echo all input redirection and shell file input to the output. If output redirection is also in effect then the input characters are also redirected to the output file. In addition, each line read from a shell file is echoed to the screen just prior to execution, but not placed in the redirected output stream. This is useful when running shell files to see that argument substitution is occurring as intended.

If you are attempting to capture a program's output into a file, to create the program documentation for example, then the Verbose Flag should be ON. If you're running a filter which reads its input and writes out characters, then you probably don't want the Verbose Flag on (or your input will be mixed with the output in the output stream.)

MicroShell automatically handles turning OFF the Verbose Flag when a pipe is invoked by the user unless overriden. See Section 2.7.1.

2.4.12 X Flag: eXit MicroShell

The X Flag permits the user to exit MicroShell back to CP/M. A warm start of CP/M occurs leaving the user at the CP/M prompt. Any multiple commands following a "-x" on a line are ignored.

See Section 4.8 on shell files for a method of exiting MicroShell to execute a CP/M command and then automatically reloading MicroShell. This is useful for running large programs which need the space used by MicroShell.

2.5 Output Redirection:

Output redirection is the process of redirecting a program's output from the console (i.e. the user's terminal) to a file. This capability is one of the UNIX features which MicroShell provides. For example, the CP/M command:

```
stat *.*
```

would normally send the CP/M utility "stat" output to the console. Under MicroShell, if the user types the command:

```
stat *.* > filename
```

MicroShell will redirect the output of "stat" to the file "filename". White space, blanks or tabs, are optional between the ">" redirection symbol and the filename. Using output redirection, the output of any program which runs under CP/M may be captured in a file for later use, editing, printing, etc. Creating documentation for a program is now much easier and more accurate since the exact screen output can be saved in a file for later incorporation into a manual. Or consider the problems of debugging a program which sends special characters to the CRT (e.g. cursor positioning, screen-clear, etc.). It's often hard to tell what's happening if the wrong special characters are being sent. With MicroShell, just redirect the output to a file and then look at the file with an editor or DDT to see what characters were sent.

2.5.1 Appending Output to a File:

To append output to an existing file, a modification of the basic output redirection function is also provided. If a user desires to place the output from a program at the end of an existing file, he types the following command line to MicroShell:

```
progname >> filename
```

MicroShell places the output of the program "progname" at the end of the file "filename". White space, blanks or tabs, are optional between the ">>" redirection symbol and the filename.

The append function is one of the features which requires the SH.OVR overlay file. If MicroShell can't find the overlay file it will sound the bell and display the error:

```
B(0) dir >> alldir
Requires SH.OVR
B(0)
```

MicroShell finds the end of the file to be appended to in one of two ways:

- MicroShell reads the last sector of the file from the disk and finds the first control Z in the last sector. If found, this is used as the end of file and appending begins where the control Z was.

- If no control Z is found in the last sector, MicroShell assumes the entire last sector is valid data and begins appending to the beginning of the

next sector.

Note that if the file contains multiple control Z's (such as when the file has been created by redirecting console output containing control Z's as screen clear characters), MicroShell may improperly find the end of file. The file will need to be purged of "extra" control Z's before appending to it.

2.5.2 Echoing Redirection to the Console:

Both output redirection (">") and appending (">>") may be echoed to the console by following the output redirection symbol with a plus sign ("+"). Thus, the command:

```
stat *.* >+ filename
```

would result in MicroShell placing the output from "stat" in the file "filename" with a simultaneous echo of the output to the console (i.e. a normal console display.) This permits the user to see what MicroShell is putting into the file and to respond to program prompts if the program requires inputs from the user.

2.5.3 Redirecting Console Output to the Printer:

To redirect console output to the printer, MicroShell recognizes a special output redirection filename -- "$P" or "$p". If the user types:

```
B(0) dir >$P
```

MicroShell sends the console output from "dir" (the list of files in the directory) to the printer instead of to the console. This is somewhat similar to CP/M's control P function. The control P function acts to send all subsequent console output to the printer and to the console, until another control P is entered. So if the user typed control P and then "dir", the "dir" output would be sent to the printer and the console as well as the prompt following the "dir" listing. Using the ">$P" feature instead of control P keeps the prompt from going to the printer and allows the user to control whether the output is also echoed to the screen. If a "+" is entered after the ">" -- i.e. "dir >+$P", then the output is echoed to the screen.

Many users didn't like the "$P" convention we chose and preferred, variously, "lst", "lst:", "lpr", etc. So we have allowed you to patch your favorite printer name. See Section 5.2.

2.5.4 Redirecting Printer Output to a File:

Various situations arise where it is desirable to be able to redirect the characters going to the printer to a file instead. Examples are accounting programs which produce some report only to the printer, dBase II reports, WordStar printer output, etc. MicroShell will redirect all output which a program is sending to the printer to a file instead with the command:

```
B(0) program >* prtfile
```

where "program" is any program which produces output to the printer, and

"prtfile" can be any filename into which to redirect the output. The "+" sign, if added, causes the output to be echoed to the printer in addition to going to the file -- e.g.:

```
B(0) program >*+ prtfile
```

Redirection of console output and printer output cannot be done simultaneously.

2.5.5 Cautions in Redirecting Output:

1. If output containing control characters is redirected, some care must be exercised with control Z's (1A Hex). Since a control Z is used by CP/M and many CP/M programs to mark the end of a text file, the first occurrence of a control Z is usually considered to be the end of the file. To most of the editors and word processors around, the rest of the file just isn't there. So if output containing control Z's is redirected to a file, a simple utility can be written as a filter to remove the control Z's from the file. Note that the filter will have to be used with the Transparency Flag on and the Mode Flag set to CP/M to allow the user to enter a unique character (like "~") at the end to exit the filter. Then the end of the file may have to be cleaned up with an editor.

2. When redirecting a program's output to a file, the size of the file often grows faster than one would expect. MicroShell will issue an error message if the disk fills up while it is redirecting output:

```
Disk Full
```

3. MicroShell will redirect console output from all sources, i.e. BDOS calls 2, 6, and 10 and direct BIOS calls. If a programmer wants to send output to the console and not to the file when output redirection is in effect, Appendix D should be consulted.

4. Input that the user types into a program whose output is being redirected is sent to the file only if the Verbose Flag is on. See Section 2.4.11.

2.6 Input Redirection:

Input redirection is the process of redirecting a program's input from the console (i.e. keyboard) to a file. For example, the CP/M command:

```
ed filename
```

would normally require the user to enter various editing commands from the keyboard. If the user is making identical changes to a number of different files, entry of the editing commands from the keyboard is not only repetitious but also error-prone. Under MicroShell, the user can place the proper editing commands in a file, say "edscript", and cause "ed" to take its input from the file "edscript" by typing the command:

```
ed filename < edscript
```

Any program which requires keyboard input can therefore take its input from a file instead of the keyboard.

2.6.1 Redirecting Console Status:

Since some programs make a console status call to CP/M before calling for input data, MicroShell includes a provision for returning "character ready" to a program. If the user types:

```
progname <- inputfil
```

MicroShell sees the "-" as a command to return "character ready" on program requests for console status. This has been made a separate command in MicroShell because some programs sample "console status" as a means of determining if the user is attempting to interrupt or abort the program (e.g. "type", "dir", "ddt", etc.) In these cases the user would not want to redirect status with the minus sign "-".

An extended shell command, Section 4.9.4, is provided to set character status true in a shell file.

Some programs cannot process input characters as fast as MicroShell can supply them when input redirection is in effect. These are typically screen-oriented programs which have to redraw the screen or issue cursor control commands after various input characters. MicroShell has a feature called "Input Ignore Factor" which causes MicroShell to send a character during input redirection and then send a number of "no character ready" status returns before sending the next character. This gives the program time to process each character before the next is received. See Section 2.4.2.

2.6.2 Normal Termination of Input Redirection:

When redirecting input to a program, normal termination will occur when the program completes its action and exits back to CP/M. This exit will occur in one of five ways:

1. Program executes a return instruction. This is what programs do that are designed not to overlay the CP/M CCP. Without MicroShell, when they exit, CP/M prompts for the next command without doing a warm start. Under MicroShell, the MicroShell prompt occurs (after MicroShell closes any output redirection files) if the program was executed as a single command. If the command line included more commands to execute (";" multiple commands used), then MicroShell loads and executes the next program. If MicroShell is executing a shell file (Section 4.0), then the next command in the shell file is executed.

2. Program jumps to location 0 which is the warm start entry point for CP/M. Without MicroShell, CP/M normally does a warm start before prompting for the next command. Under MicroShell, action is the same as in #1 above; no warm start occurs.

3. Program calls CP/M "System Reset" BDOS call (0). Without MicroShell, CP/M executes a warm start as in #2 above. Under MicroShell, action is the same as in #1 above.

4. User enters a control C as the first character in a "read console buffer" (BDOS function 10) call. Without MicroShell, CP/M executes a warm start as in #2 above. Under MicroShell, action is the same as in #1 above. If input redirection is in effect or a shell file is being executed with input from the shell file, this control C may be placed in the file as an up-arrow C ("^C") or as an actual control C (if your editor can put control characters in a file.)

5. As a final method of causing a program to exit, a special character -- %exchar -- can be defined as an exit character in an input or shell file. See Section 4.9.4 for a discussion of %exchar use.

Input redirection does not alter the exit that a program makes but the user must remember that the input file must have the same characters in it that would be typed if there were no input redirection. I.e., if a program interprets a blank line (Return only) to mean exit, then the input file must end in a blank line. If the program expects a control C or control Z, then the input file must have that control character in it. These control characters make for awkward editing (especially control Z) and if there's any other way to exit the program, it's a lot better to use it and stay away from control characters. This may seem difficult to grasp at first, but experience will make it clear.

After the program has processed all the information in an input file, what happens next depends on the state of the Mode Flag. If the Mode Flag is set for CP/M mode, input will revert to the keyboard after all characters have been used from the input file. Normal operation of the program from the keyboard then continues. This is an excellent way to give a program some initial commands and then enter an interactive mode. With WordStar, for example, the initial commands could set up various WordStar features such as margins, justification or hyphen help, and then allow the user to begin typing. There is even a provision made to take some input from an input file, shift the input to the keyboard, and then shift back to the input file. See Section 4.9.4.

The UNIX option of the Mode Flag is designed for use with programs which expect a control Z <u>followed by a carriage return</u> at the end of the input file. These are typically "filters", programs which read a stream of characters, performing some action on them, and then exit at the end of the input stream. UNICA is an example. In the UNIX mode, when MicroShell detects the end of the input file (a control Z or no more sectors in the file), a control Z followed by a carriage return is passed to the program. The program must recognize the control Z-carriage return and then terminate by one of the methods discussed above. If the program calls for more input after MicroShell has sent the control Z-carriage return, MicroShell will force an exit from the program.

2.7 Pipes:

A powerful feature of the UNIX operating system which MicroShell also provides, is the ability to "pipe" the output of one program to the input of another program. This permits building a powerful command from a series of simple tools. If the user types:

```
prog1 | prog2 | prog3 | ....
```

the output from "prog1" is sent to the input of "prog2" and the output of "prog2" is sent to the input of "prog3". The vertical bar ("|") is the command for MicroShell to create a pipeline. "Pipes" are actually a shorthand for output redirection of the first program followed by input redirection of the second program. (The UNIX user will excuse this simplification of the full UNIX inter-process communication capability.) MicroShell permits the same output and input modifiers, "+" and "-", to be used with pipelines to achieve simultaneous console echo and/or console status redirection. For example, the command:

```
prog1 |+ prog2
```

would echo the output of "prog1" to the console as it was being fed to "prog2". The command:

```
prog1 |- prog2
```

would cause MicroShell to send "character ready" status to "prog2" for its input. These two modifiers may be combined in the same pipe as:

```
prog1 |+- prog2
```

The order of the "+" and "-" is not significant.

The MicroShell "pipe" function is implemented using temporary files which MicroShell erases after the "pipe" is complete. The MicroShell pipe function is better understood by knowing that internally MicroShell actually performs a pipe as follows:

```
prog1 >pype1;prog2 <pype1 >pype2;prog3 <pype2 ...
```

The names for the temporary pipe files -- "pype#" -- were chosen to avoid conflict with a user's existing file names. Our apologies to anyone who had been using "pype1", "pype2", etc. MicroShell will erase the temporary pipe files after the pipe is successfully completed (after the input redirection step.) If a pipe does not terminate normally due to a full disk or an input redirection error, the temporary file will be left on the disk for the user to view, erase, etc. Temporary pipe files are always written to the default disk drive (i.e. the drive from which the command was initiated.) The user must insure that sufficient space exists on the drive for a temporary file containing all the output that the program being piped will generate. If the user wants to use another disk drive for the temporary file, explicit output and input redirection is necessary, e.g.:

```
prog1 >a:temp;prog2 <a:temp;era a:temp
```

2.6.1 Interaction between Pipes and the MicroShell Flags:

(If you're not a technical type, this section will no doubt bore you and you can just skip it without missing anything really important.)

A nasty CP/M convention rears its head when input redirection is used -- the end of line convention. In CP/M, lines of text or commands in files are terminated with a carriage-return, linefeed pair (0D hex and 0A hex.) For brevity, let's refer to these as CR and LF. When MicroShell is reading input from a file and passing it to a program, the program might be doing one of two things:

- Reading characters one by one and performing some action on them. This is the "filter" we mentioned above. Depending on who wrote the program, it may be expecting lines terminated by CR, LF or it may be expecting only a CR. There's no standard in this area because UNIX-like "filters" are new to the CP/M environment.

- Reading a line at a time from the input and then performing some action on the whole line. This may also be a "filter" but more commonly it is a program getting commands rather than characters to process -- an editor for example. In this case it is probably expecting a line of characters followed by a CR, just the way you would type it from the keyboard.

So we have a problem about what to do with the LF which follows the CR in our input file of data (or commands.) Strip out all the LF's or not? (UNIX by the way does not have this problem since the end of line character is specified as the "newline", a linefeed.) Previous versions of MicroShell left the user to decide what to do and set the "Gobble Line Feed" Flag accordingly. That was a problem and a nuisance!

The Solution to the Problem:

What MicroShell does now is this: whenever a pipe is invoked by the user, MicroShell assumes that a filter is being used. Not too many people actually "pipe" commands to editors, etc. We further assume that the program expects to see every character in the file, including CR's and LF's. So the Gobble Line Feed Flag is effectively turned off automatically for the duration of a pipe and then restored to its previous setting after the pipe is complete. Now we knew that this would not solve everyone's problem, so we made provision for the user to override this automatic action. See Section 5.2.

We said the Gobble Line Feed Flag was "effectively" turned off. Actually there are two other potential problems that are tied to the CR, LF problem. When the program asks for input, you normally expect to see that input echoed on the screen as you type it. Should the echo of the input go into the output file? Of course, you say. How about the case of a filter which is translating its input to upper case? You surely don't want the input characters in the output! Well, then, of course we don't want the input in the output. Well, how about the case where you're feeding commands into an program and you're trying to collect as output an example of the program actually in action. Well, you probably wouldn't use a pipe, but you might. So, to echo or not to echo.

Now the other problem -- special characters. MicroShell does a number of

things based on reading special characters (See Section 2.3.1 above.) If you are using a filter, you don't want MicroShell messing with any of the characters in the file. There's a Flag for this -- the Transparency Flag. When the Transparency Flag is on, MicroShell just reads the input file literally and passes the characters to the program, taking no action on any "special" characters.

Finally, getting to the end of a long explanation, what we do is turn off the Verbose Flag, set the Mode to UNIX and turn on the Transparent Flag. We don't have to worry about the Gobble Line Feeds Flag because if the Transparent Flag is on, there are no special characters. So pipes will automatically work right for filters, which is their most common use. If you want to use pipes in some other way, you can override this automatic setup; see Section 5.2.

2.8 Command Line Editing:

How many times have you typed in a long command only to see a mistake you made near the beginning of the line and have to erase it and start over? Or how often have you given a command only to have the program or CP/M complain about some little error? MicroShell has a built-in Command Line Editor which allows you to edit the current command line you're typing or go back and edit the previous line.

Command line editing is invoked by pressing "!" either during command entry, or at the command prompt. If the "!" is the first character on the line, the previous command line will be echoed for editing. If the "!" is not the first character, i.e. you're in the process of typing in a command and you see an error, the current command line is echoed for editing.

The "enter Command Line Editing" character ("!"), may be changed if it conflicts with normal commands you type. Note that you can never include this character in a command line you type. We could have provided an escape character for it, but there's only so much room in 8K, you know. However, you can use the Command Line Editor to put in a "!". See Section 5.2 for instructions on how to change the "enter Command Line Editing" character or any of the editing commands themselves.

Please note that this feature in no way affects CP/M's normal command line editing features; you can still use backspace, rubout (del), control R, U, and X if you want.

MicroShell's Command Line Editing works like this. After you type "!" (no carriage return is required), the appropriate command line is echoed, and MicroShell waits for the editing command. It looks like this:

```
B(0) last command line<cr>

(last command runs)

B(0) !                    <- you decide to edit the last command
Edit Mode -- Insert On    <- Edit Mode signs on with Insert Status
last command line         <- MicroShell displays
^
|_cursor is here.
```

Or, if you're typing the current command and you see an error:

```
B(0) this comand line!
          ^       ^
     error_|      |_here you type "!" to edit line

Edit Mode -- Insert On   <- Edit Mode signs on with Insert Status
this comand line   <- MicroShell displays
^
|_cursor is here.
```

Standard WordStar cursor movement and editing commands, and some additional commands, are accepted:

Command	Action
^D	- Cursor forward a character
^S or backspace	- Cursor back a character (non-destructive)
^F	- Cursor forward a word
^A	- Cursor back a word
^QD	- Cursor to end of line
^QS	- Cursor to beginning of line
^V	- Toggle insert flag (default on)
^G	- Delete character under cursor
DEL or RUBOUT	- Delete character to the left of cursor
^Y	- Delete entire line
^QY	- Delete line from cursor to the end
^Q<DEL>	- Delete line to the left of cursor
^QT	- Delete word to the left of cursor
^T	- Delete word to the right of cursor
^U	- Undo all editing and restore original line
<ESC> or ^C	- Abort editing and return to MicroShell prompt
<RETURN>	- Execute edited line

The second character of th Command Line Editing commands can be upper or

lower case or the corresponding control code. I.e. delete to end of line can be entered as ^QY, ^Qy or ^Q^Y.

The bell will sound if an attempt is made to make a command line greater than 79 characters (due to the single physical line capability of the command line editor.)

The definition of a "word" in the Command Line Editor is a little different from WordStar's rules to conveniently stop the cursor at frequently changed spots. Word separators are space, ".", ">", "<", "|", ";", and ":". These are customizable by the user. See Section 5.2.

Tabs and other control characters are not handled by the Command Line Editor; you can use them freely in your command lines and they won't bother MicroShell, but you will have some difficulty in using the Command Line Editor on those lines.

As an aside, if you are a software developer, we think command line editing is an important "user-friendly" feature of a program -- editing the input easily -- especially if complex or long lines of input are required. We will happily discuss selling the source code for this feature. Just give us a call!

3.0 Automatic Program Search

MicroShell contains two automatic search features to simplify operation under CP/M: main command (i.e. the "com" or "shell" file) search and file search.

3.1 Main Command Search:

If the user enters the command:

```
stat *.*
```

to CP/M, CP/M first looks for the program "stat.com", loads the program from the disk into main memory and then executes the program. The program to be executed will be called the "main command". CP/M will only "look" in one place for the program - on the current disk drive under the current user number. If the user is logged onto disk drive B in user number 0, CP/M will look on disk drive B for a file "stat.com" in user area 0. If CP/M does not find "stat.com" on the current drive in the current user number, CP/M will display the error message:

```
stat?
```

and prompt the user for another command.

MicroShell expands CP/M's normal search for a file to execute as follows:

1. MicroShell looks first on the current drive in the current user area for a ".com" file with the same root name, e.g. in the above example "stat.com". If this file is found, it is loaded and executed.

2. If the file is not found in step 1, MicroShell "looks" at the current user number. If the user number is greater than 0, MicroShell "looks" on the current disk drive in the user 0 area for the ".com" file. If found, the file is loaded, the user number restored from 0 to the initial user number and the program is executed. If the ".com" file is not found in user 0 (or user 0 was the initial user area), MicroShell's search proceeds to step 3.

3. MicroShell now looks at the search path specified by the "%path" extended shell function (Section 4.9.4.) The "%path" search string can specify up to 6 additional disk drives for MicroShell to check. As delivered, disk drive A is specified in the search path, but the user may change the search path to fit the needs of his system. MicroShell continues the search for the ".com" file in user area 0 of the disk located on one of the specified disk drives. If the file is found, MicroShell loads it, restores the user number and default disk drive to their initial values and executes the program. If the ".com file" is not found after searching all drives in the search path, MicroShell proceeds to the next search step.

4. Since there is not a ".com" file with the root name specified by the user on the command line, MicroShell now begins a search for a shell file with the specified root name. A shell file is MicroShell's equivalent to the CP/M SUBMIT file: a text file containing lines of commands to be executed. MicroShell builds the name of the shell file to search for by appending the command file type to the root name. In CP/M, SUBMIT files have a type of ".SUB" and this is the extension used in MicroShell as delivered. It is

specified by the extended shell function "%shext" which may be changed by the user. (See Section 4.9.4.) In UNIX, for example, shell files often have the extension ".sh". Assuming the user has not modified the shell file type, MicroShell builds a shell file name of "stat.sub" for our example above. MicroShell now begins looking for the shell file back on the current drive and user number. If found, MicroShell assumes the file contains text -- commands to be executed by MicroShell. Execution of the shell file begins. See Section 4 for more information on shell files.

5. If the shell file is not found in the current drive and user area, MicroShell's action depends on whether the user has specified automatic search for files of the command type, i.e. for ".sub" files in our case. The types of files for which MicroShell will conduct automatic searches are given by the "%ftype" exended shell function. (See Section 4.9.4.) The contents of "%ftype" are displayed in the status report ("-s" Flag.) As delivered, "%ftype" contains the type ".sub", so MicroShell will search for shell files. If "%ftype" does not contain the shell file type, MicroShell will look no further than the current drive and user for the shell file. Please note: THE ".COM" FILE AUTOMATIC SEARCH OF STEPS 1-3 ABOVE ALWAYS OCCURS AND CANNOT BE DISABLED. Even if ".com" is not in the "%ftype" list, MicroShell will still search all the drives in the search path for the ".com" file.

6. If the shell file is found in the search path, MicroShell restores the default drive and user area and begins execution of the shell file. If the shell file is not found, MicroShell displays the following error message:

```
stat?      (or whatever the command name was)
```

and prompts the user for another command.

Example: The user is initially logged onto disk drive B in user number 1 and issues the command:

```
stat *.*
```

The search path will be assumed to be the as-delivered path containing only drive A. MicroShell's search for "stat.com" proceeds as follows:

Drive	User
B	1
B	0
A	0

The user may override MicroShell's normal search path by specifying an explicit disk drive in the command. When an explicit disk drive is given, MicroShell looks first in the current user number on the disk drive specified and then in user 0 of the drive specified. For example, if the user issued the command:

```
a:stat *.*
```

from disk drive B, user area 0, MicroShell would immediately look on disk drive A, user 0 for "stat.com". If "stat.com" was not found there, MicroShell would look for a shell file "stat.sub" on drive A, user 0. If neither "stat.com" nor "stat.sub" were found, MicroShell would not look further but

would issue the error message:

a:stat?

and prompt the user for the next command. The user may want to use this feature if different versions of a program exist on two separate-drives and the user specifically wants the version on another drive.

If the user is logged onto disk drive B, in user area 1 and issues the command:

a:stat *.*

MicroShell will look first on disk drive A in user area 1 for the file "stat.com". If the program is not found there, MicroShell will look in user area 0 on drive A. If "stat.com" is not found, MicroShell will look on drive A in user area 1 for "stat.sub" and then on drive A user 0 before reporting failure.

3.2 Benefit of Main Command Search:

This feature may seem complex after reading the above discussion, but MicroShell's action is actually simple and logical; the user is freed from either prefacing commands with a disk drive or having to have a copy of all programs on every disk (and user area.) The concept of a "system disk" analogous to UNIX's "/bin" directory is not only feasible but greatly simplifies the whole operation under CP/M. If the user keeps a disk with all routinely used programs in disk drive A, he can then operate from drive B, in any user area, without concerning himself with where his programs are located. The user merely types the desired command line and MicroShell does the work of finding the program.

For single drive systems, the user can move to any user number on drive A without copying his programs to that user number.

Users with a hard disk may wish to "customize" MicroShell to reflect their system configuration (Section 5). If the hard disk is drive C, for example, the user may wish to change the as-delivered search path from drive A to drive C. Or the user could specify that drive C be checked first (after the default drive check which cannot be omitted) and then drive A. The flexibility is available to satisfy the user's needs.

3.3 File Search:

As a separate search function, MicroShell performs a search for any files which a "main command" i.e. a program, may attempt to open, If, for example, the user issued the command:

ddt file.ext

from drive B under CP/M, (without MicroShell), CP/M would load "ddt.com" from drive B and execute it. DDT would then attempt to load the file "file.ext" from drive B. If DDT didn't find "file.ext" on drive B, it would issue the error message "?" and await another command.

MicroShell "oversees" all attempts of a program to open files. If the

attempt by the program to open a particular file is unsuccessful, MicroShell "steps in" to help out the program as follows:

1. MicroShell has a list of file extensions or file types for which it is responsible ("%ftype" - See Section 4.9.4.) The file extension is ".ext" in the example above; in general, the file extension is the part of a filename after the ".". If MicroShell sees that a program was unsuccessful in opening a file, and the file extension of the file name is one for which MicroShell is responsible, and if the program didn't specify a disk drive when opening the file, then an automatic file search is performed. Now what's this "specify a disk drive" mean? In CP/M, when a program tries to open a file, there are two ways of telling CP/M where to get the file from. The program can tell CP/M exactly what drive to find the file on. In this case MicroShell assumes that the program knows what it's doing and doesn't interfere. Most of the time, however, the program "tells" CP/M just to open the file on the default drive and lets CP/M figure out what drive that is. In this case, MicroShell will step in if the open attempt fails and begin automatic searching. (For the technical types, this is the drive code in the File Control Block. Only if it is "0", meaning the default drive, does MicroShell do automatic searching. Then when MicroShell finds the file, it changes the drive code to the proper drive.) It turns out that this handles almost all of the desired situations and avoids many potential problems.

2. MicroShell uses the same search path that it uses for "main command" searching to look for the file which the user program is trying to open. Since MicroShell knows that the program was already unsuccessful in opening the file from the current disk drive and user number, it is not rechecked by MicroShell. Instead, user number 0 of the current drive is checked first if the program was running in a user number greater than 0. If the file is found in user 0, MicroShell opens it and then returns control to the program which can then read from the file just as if the file was in the current drive and user area. WRITING TO FILES USING THIS AUTOMATIC FILE SEARCH IS NOT SUPPORTED UNDER MICROSHELL AS IT CAN HAVE DISASTROUS EFFECTS. Most writing of programs we've seen is begun with a "create file" rather than an "open file", so MicroShell doesn't interfere.

3. If MicroShell does not find the file in user area 0 on the current drive, the search path specified for "main command" search is used to attempt to find the file. I.e., with the as-delivered search path of drive "A", MicroShell would look on drive A, user area 0 for the file. If found, MicroShell would open the file and return control to the user program to use the file. (Note: MicroShell changes the disk drive byte in the file control block to reflect the drive the file was found on.)

4. If after looking down the search path for the file, MicroShell cannot locate it, MicroShell returns control to the user program with the normal CP/M open failure code. The user program will then proceed to do whatever it does when it can't find the file.

3.4 Benefit of Automatic File Search:

The benefits which MicroShell provides with the file search feature are similar to those of the "main command" search; the user does not need to be concerned with where all the files that a program needs are located. He merely executes the programs as if all the necessary files were on the current

disk drive and user area and MicroShell handles the tedium of locating the files.

3.5 Some Practical Applications:

1. One popular word processing program, Wordstar, needs to access two overlay (".ovr") files for normal operation. Though Wordstar permits one disk drive other than the current drive to be searched for these files, no check across user areas is supported. Under MicroShell, the overlay files may be on the system disk, and MicroShell will find them for Wordstar from any user number or disk drive.

2. Many Cbasic application programs "chain" to subsequent ".int" files from a main menu program. MicroShell can supervise this chaining process and permit all ".int" files to be kept on the system disk.

3. The C compiler used to develop part of MicroShell, BDS C, uses two ".com" files to compile a program. After completing the first pass, the first ".com" file, "cc1.com", chains to the second ".com" file, "cc2.com", to generate the compiled code. Although a command flag for "cc1" is available to specify the drive from which "cc2" is to be loaded, MicroShell can relieve the user of this detail.

3.6 Some Practical Limitations:

There are some limitations which the user should keep in mind in using MicroShell's file search features.

3.6.1 MicroShell permits up to six disk drives to be automatically searched. The disk drive search path ("%path") can be set by the user (Section 4.9.4.) As delivered, MicroShell has only Drive A in its search path. The user should remember that each drive which MicroShell checks requires a finite amount of time: the drive must be selected and the directory read by CP/M. This time will depend on the user's particular computer system and disk drives and how many drives are checked. Therefore, the user should make the search path as short as possible while satisfying his particular needs.

3.6.2 To accomplish the automatic file search, MicroShell must compare the file extension on every failed attempt to open a file, with the list of file extensions for which MicroShell is responsible. Eight extensions are permitted to be MicroShell-responsible file extensions. In the as-delivered program, these are set to ".sub", ".ov?", and ".com" but the user may change these (Sections 4.9.4 and 5.1.) As with the search path, the user should specify only those file extensions which are necessary for his situations.

3.6.3 There will be times when the user may not want MicroShell to do any automatic file searching. MicroShell has a Flag ("-f" - Section 2.4.3) which permits the file search feature to be turned off and on from within MicroShell. Thus the user can turn it off while he runs one particular program and then turn it back on. Or he may create a shell file (Section 4) to take care of the details of this Flag and to execute a particular command. The Status Flag ("-s") may be used to determine if file search is turned off or on.

3.6.4 Some programs scan the disk directory with a special CP/M system call -- "search/search next" -- for a desired file prior to attempting to open it. If they don't "see" the desired program in the directory, they don't try to open the file. MicroShell can't help these programs because it is not possible to easily determine what file the program may be searching for (e.g. it may be using a wild character, "*" or "?", in the search.)

3.6.5 Some programs look for a file to erase before or after performing some action. Copy utilities (like CP/M's PIP) are an example of these. The user must remember that if file search is enabled and the file extension of the file being erased matches one in MicroShell's list, it will look down the search path for the file. This can have some undesirable effects if the user is not aware of what is happening. (like the wrong file gets erased!) Be careful in selecting the MicroShell-responsible extensions. A command file which turns off the file search feature, executes the desired program and then turns the file search back on is one way to solve this problem. Two specific programs are used frequently enough to warrant MicroShell automatically handling this situation: PIP and COPY. (COPY was chosen as the frequent name given to many copy programs.) MicroShell will automatically turn off the file search flag when either of these programs are run and then turn it back on (if it was initially on) after the programs terminate without any user action. If for some reason the user really wants the file search feature to be active with these features, the programs can be renamed so that MicroShell will not recognize them. MicroShell does NOT recognize "a:pip" or "b:pip", i.e. it does not check for a prepended drive name (there's only so much room for checking in an 8K program!) so get in the habit of NOT typing "pip" and "copy" with drive prefixes.

The MicroShell search features represent a powerful tool which the user can selectively configure for his needs. With the above limitations in mind, the user can customize MicroShell to most effectively meet his needs by handling repetitive, mundane details of finding programs. Customizing MicroShell is covered in Section 5.

4.0 Shell Files

MicroShell's shell file capability provides all of the functions of CP/M's "submit" feature and adds some additional features. This capability allows the user to have MicroShell execute a series of commands from a file. This file is often called either a shell file or a command file.

Sections 4.1 - 4.8 cover the basic shell file features. Section 4.9 covers extended shell file features.

4.1 Constructing the Shell File:

Using an editor, the user types the desired command lines into a file. ("pip filename=con:" may be used to more quickly create a short file. Remember to explicitly type the line feed after a carriage return which the editor normally adds automatically.)

The filename for the shell file should have an extension of ".sub" to allow MicroShell to recognize it as a shell file. This extension may be changed by the user ("%shext" -- See Section 4.9.4.)

Any of MicroShell's features may be included in the commands placed in the shell file with the following restrictions:

-T Flag: If the Transparency Flag is turned on in a shell file line, it must be turned back off in the same line. See Section 2.4.9.

-X Flag: This is the last command MicroShell will execute before exiting to CP/M.

Another shell file: If another shell file is called within a shell file (known as "nesting shell files"), MicroShell needs to save the current shell file parameters in a temporary file. The extended shell function "%sdrive" (Section 4.9.4) determines what disk drive MicroShell writes this scratch file to. It is set to drive A in the distribution version (but it may be changed by the user.) The disk in that drive must not be changed during execution of the shell file or nesting will fail. (Note: the disk must also not be marked as "Read-only" by CP/M. You can do a disk login -L at the beginning of a shell file to ensure this.) THIS TEMPORARY FILE IS USED ONLY IF NESTING OCCURS.

With these restrictions, all other MicroShell commands including redirection, pipes, multiple commands on a line, shell Flags, and CP/M commands (ERA, DIR, REN, TYPE, USER, default drive selection) may be included. MicroShell will perform normal disk drive searches for programs specified in shell files as described in Section 3.

4.2 Executing Shell Files and Argument Substitution:

To execute a shell file under MicroShell, the user types the name of the shell file, with or without the extension, followed by any arguments to be substituted in the commands in the shell file.

For example, if the file "show.sub" contained the following line:

```
type $1;stat $1
```

and was executed by typing:

```
"show doc"
```

MicroShell will substitute "doc" for all occurrences of "$1" in the shell file and execute the commands. The user can see the command lines after substitution has taken place, prior to MicroShell actually executing them, by turning on the "Verbose" (+V) shell Flag.

MicroShell accepts a total of 18 arguments, including the argument containing the shell file name ("show" in the example above.) MicroShell will in fact substitute the shell file name as typed for any occurrences of "$0" in the shell file.

If the character following a "$" in a shell file is not numeric, MicroShell will recognize that it is not an argument substitution. If there are two "$"s in a shell file, MicroShell will substitute one "$" for the two, to remain compatible with the CP/M "submit" convention.

4.3 Null Arguments:

To permit shell files to handle a varying number of arguments, no error is generated for a missing argument and MicroShell merely "closes up" the command line with the "$n" for the argument that was not specified.

For example, the Digital Research Macro Assembler, MAC, will take an optional argument specifying certain options, i.e.:

```
MAC filename $PZ
```

The "$PZ" is optional. If the shell file "domac.sub" contained the line:

```
mac $1 $2;load $1;era $1.hex
```

and was executed by typing:

```
domac test $PZ
```

then MicroShell would issue to CP/M the commands:

```
MAC TEST $PZ
LOAD TEST
ERA TEST.HEX
```

If the second argument was omitted by the user, e.g.:

```
domac test
```

MicroShell would issue the commands:

```
MAC TEST
LOAD TEST
ERA TEST.HEX
```

The space containing the "$2" in "domac.sub" would be "closed up" by MicroShell.

4.4 Control Characters in a Shell File:

If control characters are necessary in a shell file, they may be entered with an editor that will insert actual control codes, or they may be entered by the sequence "uparrow letter" (^C), where "C" is the letter associated with the control code (i.e. TAB = ^I). When MicroShell reads the line, it will substitute the appropriate control character for the ^C sequence. If a "^" is really desired in the file, it must be "escaped" with a "\" (i.e. "\^"). This substitution occurs when lines of a shell file are read and during input redirection.

4.5 Comments in Shell Files:

If a command line in a shell file begins with ":" or ";", the entire line is ignored by MicroShell in executing the shell file. This permits commenting of shell files for future edification, understanding, etc. Command lines may be "commented out" therefore by merely placing a ":" or ";" at the beginning of the line.

4.6 Input Redirection in Shell Files:

In addition to permitting normal input redirection, ("<") in a shell file command line, MicroShell provides a default redirection of input to the shell file if no other redirection of input is specified. This is analogous to the action of CP/M's "XSUB" placed in a submit file.

For example, if the shell file "disasm.sub" contained the following lines:

```
ddt $1.com >$1.asm
L100 $2
G0
```

and was executed by typing:

```
disasm test 1000
```

then the first line would cause MicroShell to load "DDT" with the file "test.com" loaded by "DDT" and output redirected to "TEST.ASM". When "DDT" called for the first line of console input, MicroShell would supply the line

"L100 1000". DDT would then output a listing of "test.com" from 100 hex to 1000 hex which MicroShell would redirect to the file "TEST.ASM". When DDT called for a second line of input, MicroShell would supply the line "GO" causing DDT to exit. MicroShell would then close the file "TEST.ASM" and return to the user with the prompt for the next MicroShell command.

This example demonstrates the power of MicroShell to make a more powerful tool by combining existing tools. The user could edit the resulting file, as desired, and reassemble it. A disassembler for free!

4.6.1 Changing the Default Input Redirection:

Input redirection to a shell file may be changed from the default input redirection from the shell file itself (which has the same effect as "XSUB" in a "submit" file) by issuing an explicit input redirection command in the shell file. In the above example, if the first line of the shell file "disasm.sub" were changed to:

```
ddt $1.com >$1.asm <script
```

then MicroShell would supply command lines to DDT from the file "script" instead of from the shell file "disasm.sub" itself. After the command with explicit input redirection terminates, MicroShell will revert back to redirecting input from the shell file on successive command lines, unless they also contain explicit input redirection commands.

The user should note that if DDT exhausts the command lines in "script" MicroShell will not shift the input back to the shell file. Some care must be used in writing files for input redirection to ensure they contain sufficient commands to cause their caller to eventually terminate. See Section 2.6.2 on terminating a program from an input file.

4.6.2 Redirecting Shell File Input to the Console:

If the user desires to be able to enter input to a program executed by a shell file from the console (keyboard), a special input file name "$T" (or "$t") is provided. Thus, in a shell file (and only in a shell file) if a command line contained:

```
ddt $1 <$T
```

DDT calls for input would be supplied from the console (keyboard). The next line which MicroShell reads from the shell file will undergo the normal input redirection process; i.e. if no explicit input direction is given, input is redirected to the shell file itself.

Some users preferred other names for the "$T" convention, so the console input filename is user-patchable; see Section 5.2.

4.7 Interrupting Shell Files:

Pressing the ESCAPE key will interrupt a shell file which is in progress. If the program currently running polls console status to see if an interrupt key has been struck, two ESCAPEs may be necessary to ensure one of them gets past the program and is seen by MicroShell.

The character recognized as the interrupt shell file character is user-programmable. In addition, the user amy disable this function completely, if desired. See Section 4.9.4.

4.8 Shell Files Which Return to CP/M:

It may be desirable to execute a command from MicroShell which requires all of the available working memory in the computer, overlaying MicroShell. This can be easily accomplished by creating a shell file which in turn creates a CP/M "submit" file and exits to CP/M to execute the "submit" file. The time overhead is not significant compared to the run time of programs which require that much memory. Here's how to do it:

Create a shell file, say "dolong.sub", which has the following lines in it:

```
a:
%print $1 $2 $3 $4 $5 >long.sub  (as many args as req'd)
%print a:sh b: >>long.sub        ("%print" is an extended shell
submit long                        function.  See Section 4.9.3.3.)
-x
```

This shell file will build a submit file, "long.sub", which "submit" will process to a "$$$.sub" file on drive A. "-x" then exits MicroShell to permit the CP/M CCP to execute the "$$$.sub" file.

To execute the shell file to do, for example, a compile with a compiler that requires all of memory, issue the command:

```
B(0) dolong compile progname arg1 arg2 arg3 arg4

  [ MicroShell builds "submit" file and exits to CP/M ]
A>compile progname arg1 arg2 arg3 arg4

  [ CP/M CCP executes "$$$.sub" file to do compile ]
A>a:sh b:

  [ CP/M CCP reloads "sh.com" as last command ]
B(0)
  [  MicroShell restarts, goes to drive B (or wherever you want, and
     waits for the next command. ]
```

Another example of a useful shell file which returns to CP/M is the following file, called "reboot.sub":

```
a:
%print a:sh b: >boot.sub
submit boot
-x
```

Execution of "reboot" will cause CP/M to warm start, reload "sh.com" and change to drive B. This may be used to login a different density disk on those systems requiring a warm start to change disk density. (For some systems, the "-L" Login Flag -- Section 2.4.5 -- will work. It depends on how the BIOS was written for a particular system.

4.9 Extended Shell Functions:

MicroShell recognizes a number of special commands beginning with a percent sign ("%") as extended shell functions and performs a particular action on them. Although the extended shell commands are most useful in shell files, they may also be executed as direct commands. To be properly recognized by MicroShell, an extended shell function must be the first, non-white space command on a line. In other words, for readability, you may have spaces or tabs at the beginning of a line.

The extended shell functions will be described in the following sections:

4.9.1 Shell Variables:

There are 33 shell variables available for use as follows:

Variable Name	Variable Type	Comments
%A - %G	Numeric	May hold values 0 - 65535, no sign (positive only.) %A and %B are local to each shell file. %C - %G are global to any shell files. Once assigned a value, that value is kept until changed or until MicroShell is reloaded.
%A$ - %G$	String	May hold variable-length strings up to a total string space of 128 bytes. A string of N chars requires N + 1 bytes. All string variables are global and retain their assigned strings until a "%clear" command or MicroShell is reloaded.
%0 - %18	String	These variables are the 19 argument substitution variables. They may be assigned values can then be passed to a program in a command line. They hold their value only until an executable program is run. Then all argument substitution variables not specified in the command line are cleared. The total space allotted to these variables is 84 chars (the max length of a command line.)

Note: The argument substitution variables are referred to as %0 - %18 in shell files when assigning values to them or using them in expressions, but are still referred to as $0 - $18 when constructing command lines. This is necessary or MicroShell would immediately try to do argument substitution on them as it read the shell file. As an example consider the following shell file which might be part of a menu program in an office environment:

```
%print -n "Enter file name to edit: "
%getstr %1 <$T
%if %1 = "" then exit
ws $1 <$T
```

4.9.2 Variable Assignment:

Shell variables may be assigned values as follows:

<var> = <constant>

<var> = <var> (Variable types must match)

<var> = <var><op><var ¦ constant> [<op><var ¦ constant> ..]

(<op> can be "+" or "-" for numeric variables but only "+" for string variables.)

Examples:

%A$ = %B$ (Assigns string in %B$ to %A$.)

%G$ = test string (Assigns "test string" to %G$. Use double quotes (") to cause leading or trailing spaces.)

%A = %A + 1 (Adds 1 to current value of %A)

%A = %A - 1 (Subtracts 1 from current value of %A)

%D = %A + %B + %C (Assigns sum of %A, %B, and %C to %D)

%C$ = %A$ + %B$ (Assigns to %C$ concatenation of %A$ and %B$.
If %A$ = "Hello " and %B$ = "world" then:
%C$ = "Hello world" after %C$ = %A$ + %B$)

%1 = %A$ (Assigns string in %A$ to %1 argument substitution variable.)

4.9.3 Shell File Programming:

Several extended shell file commands are provided to specify the flow of control in a shell file. All of these keywords begin with "%" and must be the first "non-white space" character on the logical command line so that MicroShell will recognize them.

4.9.3.1 Control Statements: The following control statements are recognized:

Statement	Action
%exit	Unconditional exit to command level
%goto <label>	Unconditional transfer of control to line with <label>
%if <expression> then <action>	Conditional transfer of control based on value of expression
%iffile <filename> <"present"|"absent"> then <action>	Conditional transfer of control based on presence or absence of file "filename". If drive specifier given on "filename", then only that drive is searched for the file "filename". If no drive specifier given, then MicroShell will perform auto-search if "filename"'s type is in the list of MicroShell-responsible file types.
%return [<assignment>]	Unconditional return to previous shell file, if nested, or command level. Optional assignment of a value to a variable is allowed, e.g.: %return %C = 1 Note: Since %A and %B are local to each shell file, assigning a return value to them would not be logical.

where:

<label> := string beginning with "%" that is not an extended shell keyword. The label must be the first non-white space on a physical (not logical) line of a shell file. The label reference in the GOTO or THEN clause need not have the preceeding "%".

Example:

```
%endless.loop
[ optional statements ]
%goto endless.loop
```

```
<expression> := <var> <op> <var> |
                <var> <op> <constant>

        <op> :=     =    (equal) |
                    #    (not equal) |
                    NE   (not equal) |
                    LT   (less than) |
                    LE   (less than or equal) |
                    GT   (greater than) |
                    GE   (greater than or equal) |
```

Example:

```
%if %A$ = stop then goto doexit
%if %A LT %D then return %C = %A
```

```
    <action> := goto <label> |
                return [<assignment>] |
                exit
```

Example:

```
%loop
[ optional statements ]
%A = %A + 1
%if %A = 6 then exit
%goto loop
```

4.9.3.2 Input Statements: The following input statements may be used to input values to variables:

Statement	Action
%getchr <var>	Get one character into variable from input. Don't forget to use "<$T" if the input is to come from the console instead of the shell file. You would almost always want the input to come from the console.

Example:

```
%print -n "Enter selection: "
%getchr %A$ <$T
```

%getstr <var>

Get a string of characters into variable from input terminated by a carriage return. Again use "$T" for input from console.

Example:

```
%print -n "Enter file name: "
%getstr %F$ <$T
```

%memnum <var>

Get numeric value from memory at 80H and place in variable named. This provides a mechanism for a program to pass a number for subsequent processing by the shell file. Note that the number must be "gotten" with %memnum before the next executable program is run since the command line for programs is stored at 80h. The number should be a two byte, binary value with the low byte at 80h and the high byte at 81h.

Example:

```
cntfile
%memnum %A
%print %A$
```

(cntfile is a user supplied program, possily in basic, which counts the number of files and stores its result at 80h.

%memstr <var>

Get a null-terminated string of characters from memory at 82H and place in variable named. This provides a mechanism for a program to pass a string for subsequent processing by the shell file. Note that the string must be "gotten" with %memstr before the next executable program is run since the command line for programs is stored at 80h.

Example:

```
calc 12.77 22.88
%memstr $A%
%print %A$
%1 = %A$
newprog $1
```

(calc is a user supplied program, possibly in basic, which stores its result as a null terminated string starting at 82h.

4.9.3.3 Output Statements: The following output statements may be used to output information from a shell file:

Statement	Action
%print [-n] [<var> ¦ <constant>]	Output the value of a variable or constant to the output stream followed by a carriage return and a line feed. (The "-n" option suppresses the carriage return line feed.) Numeric variables have leading zeroes replaced by spaces.

Example:

```
%print %A$
%print -n Strings need not be enclosed in quotes
%print -n " unless leading or trailing spaces "
%print are required.
%print -n This is;%print " one line."
```

4.9.4 Miscellaneous Statements: The following miscellaneous statements are provided. For those which have "default values", the user can change the default value permanently to suit his needs. See section 5.1.

Statement	Action
%#args <var>	Returns the number of arguments passed to a shell file. E.g., if the shell file "copy.sub" was executed by typing: B(0) copy file1 file2 then if the line: %#args %A is in the shell file "copy.sub", %A would be set to 2.
%bchar <char>	Sets the character recognized to interrupt shell files if "break" is enabled. (Default = Escape (^[))
%break on ¦ off	Enables ("on")(default) or disables ("off") the ability to interrupt shell files with the assigned break character (%bchar). If interrupt driven type-ahead is used on your system, break should be set off to prevent sampling the input during shell files.

Statement	Action
%clear	Clears shell string assignments. Does not clear argument substitution string assignment or numeric variables.
%clrstr <string>	Sets the string sent to the output to clear the screen when a %erase command is executed. Max 3 characters. (Default = ^Z)
%erase	Sends %clrstr to output to clear the screen.
%exchar <char> or %exchar none	Sets the character recognized to exit from a program in a shell file (or input redirection file) if encountered in the input stream. This would normally be used with programs which have no exit command defined or when it is not desired to go through a number of menus to exit a program. "Null" (0 Hex) is not allowed. (Default = None)
%ftype <string>	Sets file types (max 8) for which MicroShell performs auto-search, e.g.: %ftype subcomov?int Note: If less than 3-letter types are used, a space must be entered for the balance and the entire string quoted. "?" may be used for wildcarding letters in the type. (Default = SUBOV?COM)
%locase <string var>	Changes the string in the string variable specified to lower case. E.g.: %print -n "Quit? (Y/N) " %getchr %A$ <$T %locase %A$ %if %A$ = y then exit
%path <string>	Sets drive search path to drives in string, (max 6 drives), e.g.: %path abc to check first drive A, then B and finally C. (Default = A)

Statement	Action
%schar <char>	Sets the character recognized to cause the input to a program to revert from the console back to the shell file if encountered in the input stream. This would normally be used to allow the user to enter text from the console and then revert back to the shell file. "Null" is not allowed. (Default = None)
%sdrive <drive letter>	Sets the drive on which MicroShell writes a temporary file when a shell file is nested. The temporary file is 2 sectors (256 bytes) for each level of nesting. Some care must be exercised to ensure that this drive is never "read-only" (disk swap, write-protected disk, etc.)
%shext <string>	Sets the file type of shell files. (Default = SUB)
%shift	Shifts all argument substitution variables (%0 - %19) to the left. I.e. %1 -> %0, %2 -> %1, etc. Useful for sequentially processing arguments passed to a shell file.

Example: (ERASE.SUB)

```
-v;%#args %A
%print -n erase;%print -n %A;%print " files"
%loop
      %print Erasing $1
      era $1
      %shift
      %A = %A - 1
      %if %A GT 0 then goto loop
```

%status on | off	Sets character status returned to programs true if "on" (i.e. indicates that a character is always ready,) or false if "off" (default value.) Setting character status on is the same as saying: B(0) program <-infile from the command line. If status is set "on", it may be necessary to set the "Input Ignore" delay (Section 2.4.2) to prevent "swamping" the program with input.

Statement	Action
%tchar <char>	Sets the character recognized to cause the Action
%tchar <char> or %tchar none	Sets the character recognized to cause the input to a program running from a shell file to revert to the console if encountered in the input stream. This would normally be used to allow the user to enter text from the console and then revert back to the shell file later (with %schar). "Null" (0 Hex) is not allowed. (Default = None)
%upcase <string var>	Changes the string in the string variable specified to upper case. E.g.: %print -n "Quit? (Y/N) " %getchr %A$ <$T %upcase %A$ %if %A$ = Y then exit

5.0 MicroShell Customization

MicroShell, as delivered on the distribution disk, is set up for the "average" user but since many different environments are possible under CP/M, many users will want to "customize" MicroShell for their particular needs.

Two levels of customizing MicroShell are provided: Level 1 and Level 2. Level 1 customization is very easy and almost automatic. It allows the prompt, flags, search path, shell file type, shell file scratch drive, screen clear string, and the special shell file characters to be changed. In addition, Level 1 customization allows semi-automatic patching of CP/M to permit loading MicroShell automatically each time you start up your computer, IF YOUR CP/M IS CLOSE TO STANDARD.

Level 2 allows changing the number of directory columns for the "dir" command, page length for the "typ" command, multiple command separator character, printer and console redirection strings ($P and $T), Command Line Editing characters, to name a few. Level 2 customization involves the use of the CP/M debugger, DDT, or its equivalent. It requires a familiarity with DDT and some assembly language experience.

5.1 Level 1 Customization of MicroShell:

The SHSAVE program distributed with MicroShell will permit the user to change "sh.com" to reflect any changes he has made to any of the following MicroShell Flags and extended shell features:

Flags:

Command to Set	Feature	Default
+-F	File Search:	On
+-G	Gobble Line Feeds:	On
+-M	Mode - End of File(+=CP/M,-=UNIX):	CP/M
+-T	Transparency Character Flag:	Off
+-U	Upper Case Command Translation:	On
+-V	Verbose: Echo shell files:	On
+-P "string"	Prompt String:	%n%D(%u)
-D 1 NNN	Input Character Delay:	0
-D 2 NNN	Input Line Delay:	0
-D 3 NNN	Output Prompt Delay:	0
-D 4 NNNNN	Input Ignore Factor:	0

Refer to Section 2.4 for details on the function of any of the Flags.

Extended Shell Functions:

Command to Set	Feature	Default
%path "str"	Disk Drive Search Path:	A
%ftype "str"	File Types-Auto Search:	SUB OV? COM
%shext "str"	File Type - Shell Files:	SUB
%sdrive char	Scratch Drive-Shell Files:	A
%clrstr "str"	Screen Clear String:	^Z
%break on/off	Break during Shell Files:	On
%bchar char	Break Character:	Esc
%exchar char	Program Exit Character:	None
%tchar char	Revert to Keyboard Char:	None
%schar char	Revert to Shell File Char:	None

Refer to Section 4.9.4 for details on the function of any of the extended shell functions.

The user just sets up any of the above features as he wants them to be, referring to the section referenced. The status command, "-s", can be used to see the state of these features. In addition to the Flags and the extended shell functions, SHSAVE also copies the shell variable area to the new "sh.com". So if you want to preset some of the variables, just make the necessary assignments before running SHSAVE. When the user is satisfied that he has the setup he wants, he merely executes SHSAVE from MicroShell. E.g.:

```
B(0) shsave
Wrote new MicroShell (SH.COM) on drive B.
Renamed old version SHBAK.COM.
B(0)
```

SHSAVE "looks" for a copy of "sh.com" (MicroShell itself), on the disk, automatically changes all values in "sh.com" to match the values currently set and then writes a new "sh.com" on the disk. The old "sh.com" is renamed "shbak.com". Be sure there's enough room left on the disk for the new "sh.com" to be written (10K.)

SHSAVE Error Messages:

1. SHSAVE is executed as either:

```
B(0) shsave

   or

B(0) shsave [drive to write new SH.COM on]
```

If you give an improper argument, SHSAVE responds with:

```
(bell sounds)
Usage: shsave [drive]
```

2. SHSAVE must be executed from MicroShell since it copies the current status of MicroShell's variables to the new SH.COM file. If not executed from MicroShell, SHSAVE responds with:

(bell sounds)
MicroShell must be loaded to use SHSAVE

3. If SHSAVE can't find SH.COM one any of the disks in MicroShell's search path, it responds with:

(bell sounds)
Can't find SH.COM

4. SHSAVE checks the SH.COM it finds to see if it is the right version. If it's not, you probably have a pre-2.0 version of SH.COM on one of the drives in the search path. SHSAVE responds with:

(bell sounds)
SH.COM is either wrong version or bad copy

5. If SHSAVE can't write the new SH.COM to disk, it responds with:

(bell sounds)
Disk full or write protected

Make room for the new SH.COM or specify a different disk drive.

That's all there is to it! "sh.com" now becomes your customized MicroShell.

Note: Be sure not to run "shsave" on your master diskette; make a copy of it first.

The "autocpm" program will permit you to semi-automatically "patch" your CP/M system to have it load MicroShell on initial startup and run an initial command. It is "semi-automatic" because "autocpm" requires that "SYSGEN" first be run to put the CP/M system in memory. This manual step is necessary because CP/M is normally stored on the first two tracks of your diskette and the method of putting it there and getting it from there to change vary from system to system. So we made it as easy as possible. Here are the steps:

1. Copy "sysgen.com" to one of the disks in your computer.

2. Execute "sysgen" to get CP/M into memory:

```
B(0) sysgen
(SYSGEN signs on and prompts:)
Source drive name (or RETURN to skip).a          (<-- answer "a")
Source on A then type return.                    (<-- press RETURN)
Function complete.                               (<-- got system)
Destination drive name (or RETURN to terminate). (<-- press RETURN)
B(0)
```

3. Run "autocpm" with your desired initial command line. See Section 2.1 for the initial commands you can give MicroShell. For example, if you want to run MicroShell when you start up your computer and have MicroShell shift to drive B, you would type:

```
B(0) autocpm a:sh b:

AUTOCPM -- Automatic Initial Command for CP/M  (<- AUTOCPM signs on)
 Copyright 1982 New Generation Systems, Inc.

CP/M patched.  Run SYSGEN now.              (<- and reports completion)
Press RETURN for Source Drive.
For DESTINATION drive, enter drive you want new CP/M system written on.
B(0)
```

AUTOCPM Error Messages:

You may get one of the following error messages from AUTOCPM:

1. If you forget to give an initial command line after typing "autocpm":

```
(bell sounds)
Usage:   autocpm <command line to be installed>
   or:   autocpm null   (to set no auto start command)
```

Notice the syntax for entering NO initial command, i.e. taking an initial command you've put in back out again.

2. If AUTOCPM cannot find the proper "patch points" in memory:

```
(bell sounds)
AUTOCPM requires that CP/M image must be placed in
memory by SYSGEN before using. If this has been
done, and this message still appears, then your
version of CP/M is not a standard CP/M 2.2 and
AUTOCPM cannot be used to patch in a command.
```

If this message occurs, either SYSGEN did not properly read your CP/M system into memory or your CP/M system is not standard. See the comment on non-standard CP/M systems at the end of this section.

4. Run SYSGEN immediately after running AUTOCPM without doing any other operations in between. This is important since the new CP/M system is in memory awaiting the SYSGEN:

```
B(0) sysgen
(SYSGEN signs on and prompts:)
Source drive name (or RETURN to skip).              (<-- press RETURN)
Destination drive name (or RETURN to terminate).a     (<-- enter A)
Function complete.                                 (<-- wrote system)
B(0)
```

The new CP/M system is now written back out on drive A. Test that it properly runs the initial command you specified to AUTOCPM by pressing your

reset button and rebooting the system.

Non-Standard CP/M Systems:

If the new disk does not run the initial command you entered, your system manufacturer has probably modified the standard CP/M system and you will have to get his assistance in making this change. Or find a good CP/M systems programmer. There have been several articles in Microsystems magazine concerning patching CP/M to run an initial command. If you have the source code for your BIOS, and are familiar with assembly language programming, you should be able to figure out your particular system.

Note that with this initial command installed in CP/M any time you do a cold or warm boot of your system, your initial command will be executed. So, if you exit MicroShell with the "-x" Flag, the initial command will be executed again. This will reload MicroShell if the command you installed does that. In other words, you can't escape to CP/M. There are two "tricky" ways to do it, though. If you leave the disk drive prefix off "sh" when you specify the initial command to AUTOCPM, then MicroShell will only be loaded on the initial cold start of the system and whenever you exit MicroShell from drive A. If you exit MicroShell from drive B, CP/M won't be able to find "sh" (if it's not on drive B) and you'll stay in CP/M. Then it won't reload MicroShell until you do a warm start (^C) from drive A.

The other way to defeat the initial command is to erase or rename the program ("sh.com") that the initial command tries to execute. Do this in a shell file, called for example, escape.sub:

```
ren a:mysh.com=sh.com;-x
```

Now you just say "escape" and you end up in CP/M.

5.2 Level 2 Customization of MicroShell:

Level 2 customization allows changing a number of additional MicroShell features which "shsave" doesn't handle. DDT must be used to "patch" MicroShell and the "patched" MicroShell must then be "save"d. The locations for these patches are given in Appendix D.

APPENDIX A

History and Design of MicroShell

We began using CP/M in January 1978. It was a vast improvement in microcomputer operating systems; relatively easy to use, efficient in memory and disk usage and a tremendous bargain for its price. We happily used CP/M and its facilities to develop software adapting to its various features and limitations. Then in 1980 a revolutionary event occurred: we were introduced to the UNIX[tm] operating system developed by Bell Laboratories for the Digital Equipment Corporation PDP-11 minicomputer series. We were elated. Always having been shy about "big" computer operating systems and their complexity, UNIX pleasantly surprised us. It was easy to learn, easy to use and very powerful.

The user-interface to UNIX is its "shell" program, equivalent to CP/M's Console Command Processor (CCP). The idea for MicroShell really began when we used a DEC VAX minicomputer. Instead of being greeted at the terminal with DEC's operating system, VMS, we saw what looked like UNIX's "shell"! The Software Tools, which began in a book by the same name by Kernighan and Plauger and were later expanded by Lawrence Berkeley Laboratory at the University of California, had been installed "on top of" DEC's operating system. The Software Tools include a UNIX-like "shell" and give the appearance of running UNIX without losing compatibility with the native operating system or requiring the development of a whole new operating system for the VAX.

The idea was born! Why not develop a "shell" to lie "on top of" CP/M?! And in the spring of 1981, MicroShell was thus conceived.

MicroShell Design:

It was decided to implement the best functional features of the UNIX "shell" in MicroShell. The initial language chosen for development of MicroShell was "C" - the language developed by Bell Labs for writing the UNIX operating system. The Software Tools had been developed in RATFOR, a structured preprocessor for FORTRAN, which resulted in good transportability of the Software Tools from one operating system and computer to another. It was decided that the code generated by RATFOR was too large for the limited CP/M environment. In addition, transportability was a secondary goal. So BDS C was chosen as MicroShell's language.

By June 1981, MicroShell in "C" was up and running and in daily use with CP/M. At 12K, MicroShell still was larger than desired. So a rewrite of portions of MicroShell into assembly language was begun. The current version of MicroShell is written entirely in assembly language and requires about 8K bytes. Overlays are used for functions which are not time-sensitive. This has allowed expanded error messages, extended shell functions and command line editing to be included without impacting on the resident memory requirements of MicroShell. The BDS C package is still used for management of the 90+ individual modules, linking and overlay management. This package (BDS C) represents an excellent assembly-language development environment.

MicroShell is executed from CP/M by typing "sh". CP/M loads in MicroShell which then relocates itself below CP/M (just below the Basic Disk Opera-

ting System - BDOS). MicroShell replaces the CP/M Console Command Processor (CCP) and performs all of the functions of the CP/M CCP plus additional UNIX-shell-like functions. It remains resident during execution of programs until it is deliberately exited by the user. In this respect it is similar to Wordstar and other programs which themselves perform CCP functions while remaining resident.

Appendix B

MicroShell Error Messages

(Filename)?

MicroShell cannot find the file "Filename" to execute along its search path. See Section 3 for an explanation of MicroShell's search path.

CP/M Error Messages

The CP/M Basic Disk Operating System (BDOS) issues various other messages, beginning with:

BDOS ERROR ON (Drive): .....

See the Digital Research CP/M documentation for the meaning of these errors.

Error: Can`t open the input file.

Occurs during input redirection. MicroShell can't find the file specified on the command line as the input redirection file.

Error: Can`t open output file

Occurs during output redirection. MicroShell can't open the output redirection file. The directory is probably full; remove some files from the directory and repeat the command.

Disk Full

The disk has filled up while MicroShell is redirecting output (or during the first command in a pipe.) The command is aborted. Make room on the disk for the output redirection file or specify a disk other than the default disk on the command line, e.g. "stat *.* >+a:statout". The disk for the temporary pipe files ("PYPE1", etc.) cannot be specified; it is always the default disk. Use output redirection to a temporary file instead of a pipe if another disk is desired for the temporary file.

Error: More than one output file.
Error: More than one input file.
Error: Output file and pipe.

Two output or input redirection files were specified. The following are examples of <u>illegal</u> commands:

```
B(0) stat *.* >statout >statout1      (Two output files)
B(0) ed test <script <edscript        (Two input files)
B(0) type script ¦ ed test <script    (Two input files in 2nd half of
                                       command.)
B(0) type script >outfile ¦ ed test   (Two output files in 1st half of
                                       command.)
```

Note: Pipes are actually an output redirection of the first half of the pipe to a temporary file ("PYPE#") followed by an input redirection from that temporary file in the second half of the pipe. An input redirection in the first half or an output redirection (or another pipe) in the second half of the pipe are legal.

Error: Missing command name.

There was no command name where MicroShell expected one. E.g. after a semi-colon in a command line, after a pipe symbol, etc.

Error: Redirection file name missing

There was no file name given after a ">" or "<" redirection symbol.

Error: Read error on SHTMP

MicroShell couldn't read the file SHTMP which is the scratch file used when a shell file is nested. The disk specified as the shell file scratch drive (%sdrive - Section 4.9.4) must not be changed when shell file nesting is used.

Requires SH.OVR

A function was called which requires the MicroShell overlay file, SH.OVR, and the file could not be found. The overlay file is required for the append function (">>"), the MicroShell status report ("-s"), all extended shell functions beginning with "%" (e.g. %if... , %print... , etc.), and command line editing. Either the ".ovr" file type must be in the automatic search file type list (%type), or SH.OVR must be on the disk drive you are logged in on. Some error messages are shortened to a number as shown below when the MicroShell overlay file is not present.

```
Requires SH.OVR: Error #0     --> More than one output file.
Requires SH.OVR: Error #1     --> More than one input file.
Requires SH.OVR: Error #2     --> Output file and pipe.
Requires SH.OVR: Error #3     --> Can't open the input file.
Requires SH.OVR: Error #4     --> Missing command name.
Requires SH.OVR: Error #5     --> Redirection file name missing
Requires SH.OVR: Error #6     --> Read error on SHTMP in shfile
```

Can't find label: [Name of Label]
Shell file syntax error:
Line: NNNNN ->[line with error is printed]

A "goto" on line NNNNN referenced a label which could not be found. Labels must be the first word on a physical line of the shell file. An extended shell keyword (e.g. print, goto, return, clear, exit, etc.) cannot be a label. Upper/lower case is significant on labels.

Shell file syntax error:
Line: NNNNN ->[line with error is printed]

A syntax error in an extended shell file function occurred in a shell file being run. The error is on line "NNNNN" of the shell file. See Section 4.9. Typical errors are improper number of arguments in a statement, (e.g. an %if statement without a then clause), mixing variable types (assignment of a numeric variable to a string variable), variable letter out of range (variables go from A to G, remember the old ABC jingle: "ABCDEFG"), etc.

String too long.
Shell file syntax error:
Line: NNNNN ->[line with error is printed]

A string assignment in a shell file exceeded the 128 total bytes of shell string storage. See Section 4.9. "%clear" can be used to clear all shell string variables.

APPENDIX C

MicroShell Compatibility with other Programs

Compatible Programs: Many popular CP/M compatible programs have been run under MicroShell with satisfactory operation. The following is a partial list:

1. CP/M utilities: ASM, DDT, ED, LOAD, STAT, PIP, SYSGEN
2. Assemblers: MAC, ACT 80, ACT 86
3. Word Processors: Wordstar/Spellstar, Benchmark, Spellbinder, Mince
4. Languages: BDS C, CBASIC II, BASCOM, PASCAL/M
5. Apple CP/M (with Microsoft Softcard)

Incompatible Programs: Programs which are incompatible with MicroShell are usually accessing information inside of CP/M rather than using the normal CP/M entry points. MicroShell depends on a program using the normal entry points for CP/M: the BDOS entry point at location 5 and the warm start entry point at location 0.

The following programs are known to be incompatible with MicroShell:

1. CP/M utilities:

   MOVCPM - Requires that the CP/M CCP be present. Exit MicroShell to do MOVCPM.

   SUBMIT - MicroShell provides a capability equivalent to the CP/M SUBMIT function. SUBMIT itself requires that the CP/M CCP be present. A useful feature is described in Section 4 for having MicroShell create a submit file and then exit to CP/M to execute the submit file. This permits long programs, which require all of the computer's memory, to be started from MicroShell and MicroShell reloaded automatically on completion of the program.

2. CP/M User Group Programs

   Some of the earlier utilities in the CP/M Users' Group did not access CP/M via its design entry points. Some of these programs may not operate correctly with MicroShell.

APPENDIX D

Customization Locations

Several MicroShell features are patchable using the CP/M debugger, DDT, or its equivalent. We will not describe the use of DDT. See the Digital Research CP/M documentation for a discussion of how to use DDT.

1. Variables in SH.COM which can be patched. Use "SAVE 38 SH.COM" after all patches are complete.

Variable	Default	Location	Comment
";"-Multiple Command Separator	";"	0FEAH 1A18H	Both locations must be changed. Be sure new char does not interfere with any MicroShell special chars. (Use "/")
"PYPE" - Pipe File Name	"PYPE"	11B5H	4 chars followed by 0. (Upper case.)
$P printer string	$P	11BAH	5 chars followed by 0. (Upper case)
$T terminal string	$T	11C0H	5 chars followed by 0. (Upper case)
Columns in "DIR"	4	251BH	Set as desired
Lines/page for "TYP"	23 (17H)	2523H	Set to your lines/ screen - 1
Edit Command Character	! (21H)	252EH	Set to character to enter edit mode
Auto Pipe Flag Set	0	25CBH	0 = Transparent On 1 = No auto set.

2. Variables in SH.OVR for WordStar-like Command Line Editing which can be patched. Use "SAVE 38 SH.OVR" after all patches are complete.

"Word Separators" for "next word", "last word", etc. functions:

```
2028H   cpi  20H
        rz
        cpi  2Eh
        rz
        cpi ....  ;and so on

        ret       ;patch a NOP here
        cpi  20H  ;add your word separators here
        rz
        cpi  20H  ;and here, etc.
        rz
```

Insert Mode Toggle:

```
2358H     db    0FFh ;0FFH = ON, 00 = Off
```

Command Dispatch Table for Editing Commands:

```
2366H     db    1st char,2nd char   ;2nd char = 0 for 1 char commands
          dw    routine address

          db    etc....
          dw    ....

23CEH     db    0                   ;end of table
```

3. Structure of MicroShell for Programming Interface:

```
0005      jmp   base        ;MicroShell Intercept (Add 1006H if you
                            ;look at 6H using DDT

base      jmp   hook        ;jump to MicroShell routine which
(XX06H)                     ;determines what BDOS call is being made
                            ;and vectors it as appropriate
base+3    jmp   BDOS        ;here is a jump to the real BDOS
                            ;call it for "getting past" MicroShell


BDOS-012AH:                 ;table of original BIOS jumps before
          jmp   coldboot    ;MicroShell overlays BIOS jump table
          jmp   warmboot
          jmp   const
          jmp   conin
          jmp   conout
          jmp   list
          jmp   punch
          jmp   reader
          jmp   home
          jmp   seldsk
```

Appendix E

BIBLIOGRAPHY

The following books and articles represent a few of the sources of information on the UNIX operating system and its Shell.

Bourne, S. R., "An Introduction to the UNIX Shell." The Bell System Technical Journal 57 (1978):2797-2822.

Gautier, Richard L. Using the UNIX System. Reston Publishing Company, Inc. A Prentice-Hall Company, 1981.

Hall, D. E.; Scherrer, D.K.; and Sventek, J. S. "A Virtual Operating System." In Communications of the ACM 23 (1980):495-502.

Johnson, Stephen C. "UNIX Time-Sharing System: Language Development Tools." Bell System Technical Journal 57 (1978): 1971-90.

Kernighan, Brian W., and Plauger, P. J. Software Tools. Reading, Massachusetts: Addison-Wesley, 1976.

Kernighan, Brian W., and Ritchie, Dennis M. The C Programming Language. Englewood Cliffs: Prentice-Hall, 1978.

Thomas, Rebecca and Yates, Jean User Guide to the UNIX System. Berkeley, California: OSBORNE/McGraw-Hill, 1982.

Appendix F

Summary of MicroShell Commands

Special MicroShell Characters in Command Line

Char	Meaning	Example
> >*	Output Redirection OF console Output Redirection OF Printer	stat >filename pip lst:=file >*prntfile
<	Input Redirection	ed file <script
|	Pipe output to input of next cmd	prog1 | prog2 | ... stat *.*|pip lst:=con:
^	"^" in shell and input files causes next character to be its control equivalent.	^C (or ^c) changed to 03
: ;	When first character on a shell file line, causes line to be treated as a comment (ignored).	: this is a comment ; this is a comment too
+	Echo redirected Output to Console	stat >+filename prog1 |+ prog2 | ...
-	Return "character ready" to console input status calls	sysgen <-script prog1 |- prog2 | ... prog1 |+- prog2 | ...
;	Separate commands	era *.bak;stat;ed test <script
"	Treat arguments with embedded spaces or tabs as 1 argument and ignore special characters inside quotes	echo "This is one argument"
\	Ignore special meaning of next character	dir \>file (filename ">file")
$	Argument substitution in shell (command) files (0-18)	copyfile test if "copyfile.sub" contains: pip b:=a:$1 then MicroShell executes: pip b:=a:test
!	WordStar-like Command Line Editing	B(0) mustake!
$T $t	Redirect Input back to console in a shell file	ddt <$T
$P $p	Redirect Output to Printer	dir >$P

Shell Flags

Flag	Meaning	Example
+f or +F (Default)	Auxiliary file search enable	B(0) +F (Auxiliary file search on)
-f or -F	Auxiliary file search disable	B(0) -F (Auxiliary file search off)
+g or +G (Default)	Gobble line feeds during Input redirection	B(0) +G (Line feeds removed from input)
-g or -G	Don't gobble line feeds during Input Redirection	B(0) -G
-l or -L or +l or +L	Login current disk (after changing disks)	B(0) -l B(0) (new disk logged into CP/M for writing)
-p or -P or +p or +P	Prompt string Uses "C"-like format: % - Next char special (%% gives %) n/N - Newline (CR, LF) d - Lower case drive D - Upper case drive u/U - User number	-p "%n%D(%u) " gives: (CR, LF)B(0) -p "%nDrive:%D User%U %" gives: (CR, LF)Drive:A User:0 %
-s or -S or +s or +S	Shell Status report (Shows status of flags)	B(0) -S [displays status report]
+u or +U (Default)	Upper case translation of command line (like CP/M)	B(0) +U B(0) echo this is upper case THIS IS UPPER CASE
-u or -U	No case translation on command line (allows passing lower case command line to a program)	B(0) -U B(0) echo this is lower case this is lower case
+v or +V	Verbose mode: Echo commands before execution	B(0) +V (All commands echoed) comfile test data pip b:=a:test pip b:=a:data
-v or -V (Default)	Disable Verbose mode	B(0) -V (No echo of commands)
-x or -X	Exit MicroShell and return to CP/M	B(0) -x B>

Additional Flags: D - Section 2.4.2, M - Section 2.4.6, T - Section 2.4.9

APPENDIX G

INDEX